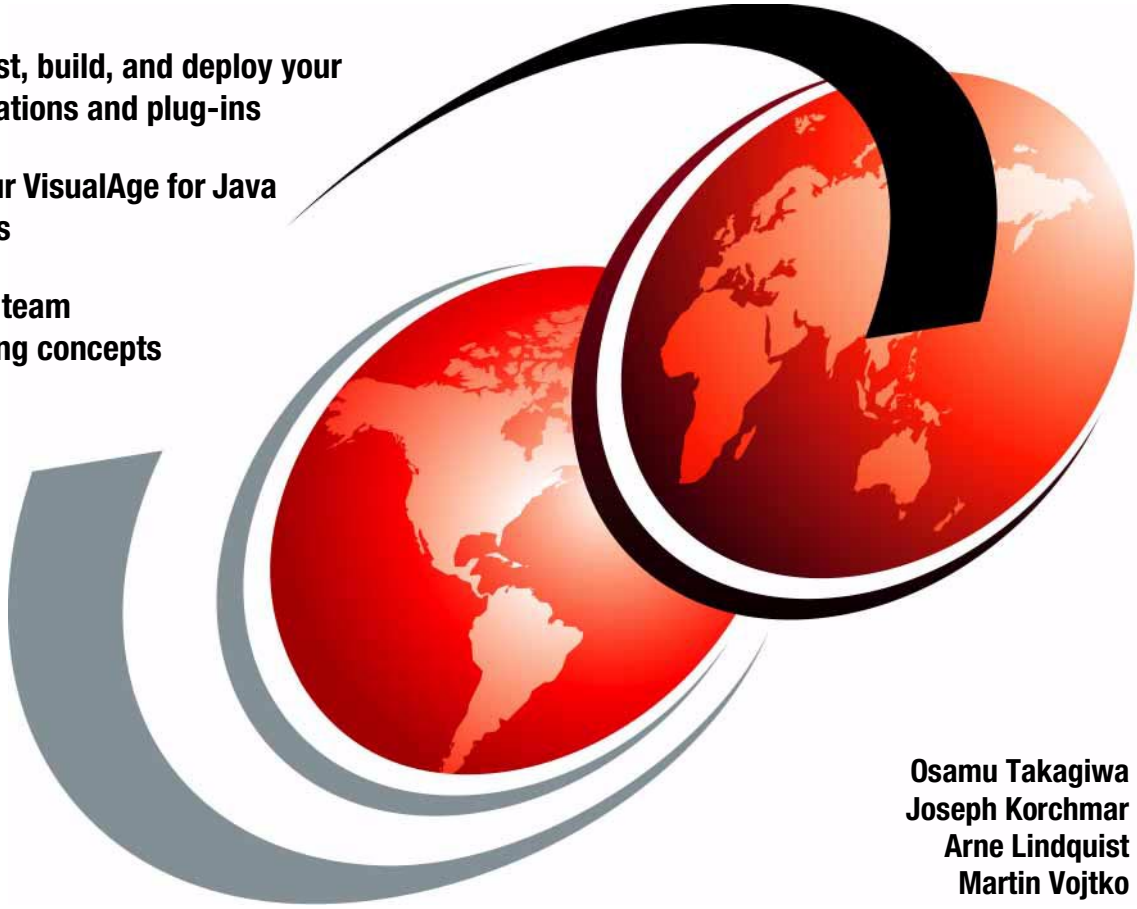
WebSphere software

IBM

WebSphere Studio Application Developer

Programming Guide

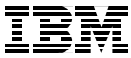
- Develop, test, build, and deploy your Web applications and plug-ins
- Migrate your VisualAge for Java applications
- Experience team programming concepts



Osamu Takagiwa
Joseph Korchmar
Arne Lindquist
Martin Vojtko

Redbooks

ibm.com/redbooks



International Technical Support Organization

WebSphere Studio Application Developer Programming Guide

August 2002

SG24-6585-00

Take Note! Before using this information and the product it supports, be sure to read the general information in “Notices” on page xvii.

First Edition (August 2002)

This edition applies to Version 4.0.2 of IBM WebSphere Studio Application Developer for use with Microsoft Windows.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002. All rights reserved.
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Tables xv

Notices xvii
Trademarks xviii

Preface xix
The team that wrote this redbook xx
Notice xxi
Comments welcome xxii

Part 1. Introducing WebSphere Studio 1

Chapter 1. Concepts 3
1.1 Introduction 4
1.2 Eclipse 4
1.3 Workbench architecture 5
1.4 Workbench features 7
1.4.1 Plug-in based tooling 7
1.4.2 Role-based development with consistent look and feel 8
1.4.3 Vertical and horizontal integration 8
1.4.4 Open standards 8
1.4.5 Open team development 9
1.4.6 File-based IDE 9
1.5 WebSphere Studio product family 9
1.6 Tools 13
1.6.1 Web development tools 13
1.6.2 Relational database tools 14
1.6.3 XML tools 14
1.6.4 Java development tools 15
1.6.5 Web services development tools 16
1.6.6 EJB development tools 16
1.6.7 Team collaboration 16
1.6.8 Debugging tools 17
1.6.9 Performance profiling tools 17
1.6.10 Server tools for testing and deployment 17
1.6.11 Plug-in development tools 18
1.7 WebSphere Studio V4 and VisualAge for Java 19
1.8 Sample code in this book 19
1.9 Sample plug-ins in this book 21

© Copyright IBM Corp. 2002

Chapter 2. Setting up your workbench 23
2.1 Defining Java class path variables 24
2.2 Java coding preferences 25
2.2.1 Code formatting. 25
2.2.2 Java editor preferences. 26
2.2.3 Import organization 27
2.2.4 Refactoring 28
2.2.5 Choice of JRE 29
2.3 Automatic builds 30
2.4 Workbench window preferences 31

Chapter 3. Perspectives, views, and editors. 33
3.1 Integrated Development Environment (IDE) 34
3.1.1 Perspectives 34
3.1.2 Views. 34
3.1.3 Editors 35
3.2 Resource perspective 37
3.2.1 Hierarchy levels. 38
3.2.2 Changing the default perspective 39
3.3 Opening another perspective 40
3.4 Java perspective 43
3.4.1 Type Hierarchy view 45
3.4.2 Java code editor 48
3.5 Web perspective 52
3.5.1 Page Designer 54
3.5.2 Web.xml editor 59
3.6 J2EE perspective 61
3.6.1 Application editor. 63
3.6.2 Application Extension editor 66
3.6.3 EJB editor 68
3.6.4 EJB Extension editor. 72
3.7 Server perspective 74
3.8 XML perspective 76
3.8.1 XML editor. 78
3.8.2 DTD editor. 80
3.8.3 XML schema editor 81
3.8.4 XSL trace editor 83
3.8.5 XML to XML mapping editor 83
3.8.6 RDB to XML mapping editor 84
3.9 Data perspective 85
3.9.1 SQL Query Builder 90
3.10 Debug perspective 91
3.11 Profiling perspective 93

3.12 Script perspective 96
3.13 Team perspective 100
3.14 Help perspective 103
3.14.1 Workbench Help tools 103
3.14.2 Application Developer Online Help 104
3.15 Tasks view 106

Chapter 4. Application Developer terminology 111
4.1 J2EE architecture 112
4.2 Projects 113
4.3 Project organization 117
4.3.1 Enterprise project organization 117
4.3.2 Java project organization 118
4.3.3 Web project organization 118
4.3.4 Application client project organization 120
4.3.5 Server project organization 120

Chapter 5. Programming assists 121
5.1 Pluggable JDK 122
5.2 Java snippets 122
5.3 Code Assist feature 123
5.4 Import generation 125
5.5 Tasks view 127
5.6 Refactoring 127
5.7 Smart compilation 133
5.8 Java search 133
5.9 Bookmarks 134
5.10 Integrated debugging 136

Part 2. Developing applications 137

Chapter 6. Creating Java applications 139
6.1 Java applications 140
6.2 Creating a Java project 140
6.3 Create Java packages and classes 146
6.4 Running your code 149
6.5 Locating compile errors in your code 156
6.6 Debugging your code 157
6.7 Java development in Application Developer 159

Chapter 7. Creating HTML resources 161
7.1 Creating a Web project 162
7.1.1 Web project directory structure 165
7.2 Working with Page Designer 166

7.2.1 Create a simple HTML page 167
7.2.2 Creating an HTML page with a form 171
7.2.3 Linking to another HTML page 175
7.3 Creating and using a Java applet 177
7.4 Creating Web pages from a JavaBean 182
7.5 Importing an existing Web site 193
7.6 Creating and using graphics 196
7.7 CSS File Wizard 198

Chapter 8. Creating Web applications with dynamic content 203
8.1 Wizards 204
8.2 Working with servlets 204
8.2.1 Create or open your Web project 205
8.2.2 Adding a servlet to your Web project 205
8.2.3 Creating a simple servlet. 207
8.2.4 Editing the servlet 210
8.2.5 Testing the servlet. 213
8.2.6 Conclusion 217
8.3 Model-View-Controller pattern. 218
8.3.1 Model 219
8.3.2 View 219
8.3.3 Controller. 219
8.3.4 MVC usage rules. 220
8.3.5 The MVC pattern in Web applications. 222
8.3.6 Extending the MVC pattern to distributed applications 222
8.4 Developing Web applications using the MVC pattern 224
8.4.1 Creating the model Java bean 224
8.4.2 Creating the view JSP. 230
8.4.3 Creating the controller servlet 247
8.5 Testing your Web application 253

Part 3. Database applications 259

Chapter 9. Database connectivity 261
9.1 JDBC overview 262
9.2 Data source versus direct connection 263
9.3 Application Developer database operations 265
9.4 XMI and DDL. 266
9.5 Data perspective 266
9.6 Using DB Explorer. 269
9.6.1 Creating a database connection 270
9.6.2 Import database objects 273
9.6.3 Generate DDL and XML Schema files 274
9.7 Creating database objects. 275

Chapter 10. Using SQL Wizard and SQL Query Builder 283
10.1 Using the SQL Wizard 284
10.2 Using SQL Query Builder 290

Chapter 11. Stored procedures 297
11.1 What is a stored procedure? 298
11.2 Creating a Java stored procedure 298
11.3 Accessing a Java stored procedure 299

Chapter 12. Accessing databases from your applications 301
12.1 Accessing databases from a Java application 302
12.2 Accessing databases from a Web application 303
12.2.1 Generate Web pages from SQL queries 304
12.2.2 Accessing a database using DB Beans 315
12.2.3 Accessing a database using JSP taglib 316

Part 4. Migrating your applications 323

Chapter 13. Migrating your Java classes 325
13.1 Importing Java classes 326
13.2 Installing an EAR 326
13.3 Fixing import problems 330
13.4 Importing PDKLite 332
13.5 EJB specification 334
13.6 Fixing JSP tag error 336
13.7 Fixing web.xml 336
13.8 Running PDKLite 337

Chapter 14. Migrating your VisualAge for Java project 339
14.1 VisualAge for Java 340
14.1.1 What's new in Application Developer 340
14.1.2 Coexisting with VisualAge for Java 340
14.1.3 Project structure 340
14.1.4 Visual Composition Editor (VCE) 341
14.1.5 Source location 342
14.2 Migrating a Web project 342
14.3 Setting up your project 344
14.4 Importing a jar file 344
14.4.1 Fixing the class path 345
14.4.2 Inner classes 347
14.4.3 Deprecated methods 347
14.5 Running the samples 348
14.5.1 WTE configuration 348
14.6 Importing JSPs and HTML files 349

14.6.1 Changing the invoker to alias style 350
14.6.2 HTML tags. 351
14.6.3 JSP tags 351

Chapter 15. Other migration tips 353
15.1 SQLJ and stored procedures 354
15.2 VisualAge Persistence Builder 354
15.3 Enterprise JavaBeans 355
15.4 Using the CVS repository from VisualAge for Java. 357

Part 5. Testing and deploying your Web applications 361

Chapter 16. Server instances and server configurations 363
16.1 Server Tools feature 364
16.1.1 Supported run-time environments. 364
16.1.2 WebSphere Test Environment benefits 365
16.2 Server perspective 366
16.3 Creating a server project automatically 367
16.4 Manual server instances and configurations 369
16.4.1 Creating a Server project 370
16.5 Creating an instance and configuration separately 372
16.5.1 Creating a server instance manually. 372
16.5.2 Creating a server configuration manually 382
16.6 Creating an instance and configuration together. 386
16.7 Adding a project to a server configuration. 388
16.8 Replacement of default files 389
16.9 Starting a remote instance 390
16.10 Stopping remote instance 392
16.11 Apache Tomcat 392

Chapter 17. Testing and debugging your application 393
17.1 Debug perspective 394
17.2 Debugging a Java program 395
17.2.1 Setting breakpoints in the code. 395
17.2.2 Testing the application with breakpoints enabled 396
17.2.3 Debug functions 399
17.2.4 Watching variables 400
17.3 Manually debugging on a remote WebSphere AEs 401
17.3.1 Starting the remote server. 401
17.3.2 Starting Application Developer in remote mode 401
17.3.3 Disconnecting from the remote VM. 403

Chapter 18. JUnit test framework 405
18.1 What is JUnit 406

18.1.1 Unit testing 406
18.1.2 Why unit testing? 406
18.1.3 Benefits of a unit testing framework 407
18.1.4 How to test 408
18.1.5 TestCase class 409
18.1.6 TestSuite class 409
18.2 Installing JUnit 409
18.3 Creating the test case 410
18.3.1 Test case class definition 411
18.3.2 The setUp and tearDown methods 412
18.3.3 Testing body 412
18.3.4 Creating test methods 413
18.3.5 Create a TestSuite 414
18.3.6 Implementing TestRunner 414
18.4 Running the test case 414
18.4.1 Testing the servlet 416

Chapter 19. Deploying your Web application 419
19.1 Manual deployment 420
19.1.1 Exporting your project from Application Developer 420
19.1.2 Installing the EAR file on WebSphere AEs 421
19.1.3 Starting the WebSphere AEs Admin Console 421
19.1.4 Installing the EAR 422
19.1.5 Testing the Application 425
19.2 Publishing to a remote server 426
19.2.1 Creating a remote server instance 427
19.2.2 Publishing to a remote server 437
19.2.3 Testing the application 438

Chapter 20. Building your application with Ant 441
20.1 What is Ant? 442
20.2 Ant build files 442
20.3 Built-in tasks 443
20.4 Creating a simple build file 444
20.4.1 Setting global properties 446
20.4.2 Build targets 446
20.5 Running Ant 447
20.5.1 Classpath Problem 450
20.6 Working with J2EE 450
20.6.1 Ant Extra plug-in 450
20.6.2 Exporting a WAR file 451
20.6.3 Exporting an EAR file 452

Part 6. Profiling 455

Chapter 21. Profiling concepts . . . 457
21.1 Profiling architecture . . . 458
21.2 Performance analysis . . . 459
21.2.1 Class statistics . . . 460
21.2.2 Method statistics . . . 462
21.2.3 Heap . . . 463
21.2.4 Object reference . . . 466
21.2.5 Execution flow . . . 468

Chapter 22. Profiling your application . . . 471
22.1 Configuring WebSphere Test Environment . . . 472
22.2 Enabling the profiling agent . . . 472
22.3 Starting and stopping profiling . . . 474
22.4 Profiling remote processes . . . 479
22.5 Some things to be aware of . . . 480

Part 7. Team programming . . . 481

Chapter 23. Version control . . . 483
23.1 Workspace . . . 484
23.1.1 Working with the local history . . . 484
23.1.2 Configuring multiple workspaces . . . 486
23.2 Team development . . . 487
23.2.1 Team roles . . . 488
23.2.2 Team terminology . . . 488
23.2.3 Optimistic concurrency model . . . 491
23.2.4 Ideal work flow . . . 492
23.2.5 SCM integration . . . 493
23.2.6 Application Developer Team Support . . . 494
23.2.7 Terminology comparison . . . 495

Chapter 24. Using CVS . . . 497
24.1 How CVS works . . . 498
24.2 SCM perspectives . . . 499
24.2.1 Team perspective . . . 500
24.2.2 Resource Perspective . . . 501
24.3 Creating a CVS repository . . . 501
24.3.1 Installing CVS . . . 502
24.3.2 Connecting to a repository . . . 505
24.4 Local versus remote repository . . . 511
24.5 Streams in CVS . . . 512
24.5.1 Creating a new stream . . . 513
24.5.2 Viewing stream resource history . . . 515
24.5.3 Associating a project with a stream . . . 516

24.5.4 Splitting from a stream 519
24.5.5 Merging from a stream 523
24.6 Team specific actions 527
24.7 Comparing the resources 528
24.7.1 Three way compare 528
24.7.2 Comparing resources 529
24.8 Saving your work 532
24.9 Catching up your project 533
24.9.1 Incoming and outgoing changes 533
24.9.2 Dealing with conflicting changes 533
24.9.3 Merging changes 534
24.10 Versioning your project 535
24.11 Releasing your project 536
24.12 Team development simulation 538
24.12.1 Configuration 538
24.12.2 Sequential development scenario 539
24.12.3 Parallel development in a single stream scenario 542
24.12.4 Branching using multiple streams scenario 546
24.12.5 Merging streams 548
24.13 Additional team topics 550
24.13.1 Determining which files are managed 550
24.13.2 Backing up the CVS repository 550
24.13.3 Repository management 551
24.13.4 Implementing security 551
24.13.5 Build scripts 552
24.13.6 Managing class paths 552
24.13.7 Watching a file for changes 553
24.13.8 Other CVS commands 553

Chapter 25. Using Rational ClearCase 555
25.1 What is Rational ClearCase? 556
25.1.1 ClearCase highlights 557
25.1.2 ClearCase LT 557
25.1.3 ClearCase and ClearCase MultiSite 558
25.2 Installing ClearCase LT 559
25.2.1 Installing ClearCase LT Server 559
25.2.2 Installing the ClearCase LT client 561
25.2.3 Learning ClearCase LT 563
25.3 Basic ClearCase terminology 564
25.4 ClearCase and Application Developer 566
25.4.1 ClearCase help in Application Developer 566
25.4.2 Rational ClearCase preferences settings 567
25.5 Using ClearCase with Application Developer 567

25.5.1 Create a new ClearCase VOB 568
25.5.2 Associate with a ClearCase view 569
25.5.3 Move a project into ClearCase 575
25.5.4 Check in a resource 579
25.5.5 Check out a resource 580
25.5.6 The ClearCase Explorer 582
25.5.7 Conclusion 584

Part 8. The plug-in development environment 587

Chapter 26. Understanding the PDE 589
26.1 Introducing PDE 590
26.1.1 Platform subsystems 590
26.1.2 Concepts 591
26.2 Configuring PDE 592
26.2.1 Run-time instance configuration 592
26.2.2 Selecting external plug-ins 593
26.3 Setting up the workbench 593
26.4 Extension points 596
26.4.1 Plugging into the workbench 598
26.4.2 Manipulating Java code 599

Chapter 27. Adding a plug-in to the workbench 603
27.1 Creating a project 604
27.2 Creating the plug-in 608
27.2.1 Selecting an extension point 609
27.2.2 Adding a view 610
27.2.3 Implementing the ITSOJarView class 612
27.2.4 Dependencies 613
27.2.5 Testing your plug-in 614
27.2.6 Debugging your plug-in 615
27.2.7 Adding an icon 616
27.3 Listening to a selection event 617
27.4 Creating an editor 619
27.4.1 Running the new plug-in 623

Chapter 28. Creating AntTask with JDT API 625
28.1 Writing a task 626
28.1.1 Parser time and run time 627
28.1.2 Project class path generator 628
28.1.3 Using JDT API 629
28.2 Creating a task plug-in project 629
28.2.1 Creating a task class 631
28.2.2 Running a task 634

Chapter 29. Deploying a plug-in . . . 637
29.1 Publishing a plug-in . . . 638
29.1.1 Building a plug-in JAR . . . 638
29.2 Installing the plug-in . . . 641
29.3 Fragment development . . . 642
29.3.1 Writing a fragment for ITSOJarEditor . . . 643
29.3.2 Building a fragment . . . 648
29.4 Build configuration . . . 649
29.5 Creating a component . . . 650
29.5.1 Setting up a component project . . . 651
29.5.2 Creating a component project . . . 652
29.5.3 Synchronizing versions . . . 656
29.5.4 Generating a component JAR . . . 657
29.6 Publishing a JAR . . . 658

Part 9. Appendixes . . . 661

Appendix A. Installing WebSphere Studio Application Developer . . . 663
Things to do before installation . . . 664
Installing WebSphere Studio . . . 664
Verifying the installation . . . 666

Appendix B. Installing IBM WebSphere Application Server 4.0 AEs . . . 669
Things to do before installation . . . 670
Hardware and software prerequisites . . . 670
Create groups and users . . . 670
Check that IP ports are unused . . . 671
Stop the Web server processes . . . 671
Install WebSphere . . . 671
Verifying the installation . . . 673

Appendix C. Additional material . . . 675
Locating the Web material . . . 676
Using the Web material . . . 676
System requirements for downloading the Web material . . . 676
How to use the Web material . . . 677
Installing the sample database . . . 677

Abbreviations and acronyms . . . 681

Related publications . . . 683
IBM Redbooks . . . 683
Referenced Web sites . . . 683
How to get IBM Redbooks . . . 684

IBM Redbooks collections 685

Index 687

Tables

5-1 Supported refactoring operations. 128
16-1 What projects can run in which environment 365
18-1 JUnit assert methods 414
23-1 Application Developer/CVS/ClearCase terminology comparison. 495
25-1 Key capabilities of ClearCase 558
25-2 Comparison of ClearCase and ClearCase LT features 558
25-3 ClearCase functions in Application Developer - overview 585

© Copyright IBM Corp. 2002

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright IBM Corp. 2002

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
DB2®
IBM®
IMS™
Informix®
Perform™
Redbooks(logo)™ 
SAA®
Sequent®
SP™
VisualAge®
WebSphere®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM Redbook is a programming guide for the application development tool, WebSphere Studio Application Developer. This tool is not only for the Java developer, but also for the Web designer who creates Web pages. The WebSphere Studio Application Developer basic tooling and team environment is presented along with the development and deployment of Web applications.

WebSphere Studio Application Developer provides integrated development tools for all e-business development roles, including Web developers, Java developers, business analysts, architects, and enterprise programmers. The customizable, targeted, role-based approach of WebSphere Studio Application Developer will be characteristic of all new products built on the WebSphere Studio Workbench. It is well integrated with WebSphere Application Server and provides a built-in single server that can be used for the testing and profiling of Web applications. It replaces the existing Java and Web application development tools, VisualAge for Java and WebSphere Studio.

This redbook consists of eight parts: an introduction to the WebSphere Studio family of products and a sample application that is used in this book, Web page and Java Web application development, database connectivity, migrating from VisualAge for Java, testing and deploying a Web application with WebSphere Application Server, profiling, team development with Concurrent Versions System or Rational ClearCase LT, and plug-in development in WebSphere Studio Application Developer.

© Copyright IBM Corp. 2002

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.



Osamu, Arne, Martin, Joe

Osamu Takagiwa is an Advisory IT Specialist at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide in all areas of application development. Before joining the ITSO one and a half years ago, Osamu worked at IBM Japan as an IT Specialist.

Joseph Korchmar is a Technical Advisor with Wachovia Corporation in Charlotte, North Carolina. He has 19 years of experience at IBM and 4 years of experience at Wachovia. He has 10 years of experience in hardware engineering and 13 years of experience in software engineering. He holds a bachelor's degree in Computer Science with a Mathematics minor from the University of North Carolina and an Associate's degree in Electronic and Computer

Technology from the Electronic Institute of Pittsburgh, Pennsylvania. His areas of expertise include object-oriented analysis, design and programming, Java, Smalltalk, C/C++, client/server applications, DB2, WebSphere Application Server, UNIX, mainframes and workstations.

Arne Lindquist is an IT Specialist with IBM Global Services, Australia. He has 19 years of software development experience, covering both mainframe and workstations. His areas of expertise include C++, Java, Visual Programming tools, Web development, object-oriented development, and client/server applications. Arne has a degree in Accounting and Finance from the Stockholm School of Economics.

Martin Vojtko is an IT Consultant with IBM Global Services, Austria. He has 19 years of experience in object-oriented programming (workstations). He has worked at IBM for 6 years. His areas of expertise include object-oriented analysis, design and programming, Smalltalk, Java, Web development, WebSphere and VisualAge. Martin has a degree in Computer Science from Bratislava Comenius University (Slovakia).

Thanks to the following people for their contributions to this project:

Barry Searle
IBM CANADA, Toronto

Ueli Wahli
IBM USA, San Jose

Reginaldo Barosa
IBM USA, Boston

Notice

This publication is intended to help WebSphere developers learn how to use WebSphere Studio Application Developer product. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM WebSphere Studio Application Developer 4.0x. See the PUBLICATIONS section of the IBM Programming Announcement for IBM WebSphere Studio Application Developer for more information about what publications are considered to be product documentation.

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to the address on page ii.

Part 1

Introducing WebSphere Studio

Part 1 introduces WebSphere Studio Application Developer.

© Copyright IBM Corp. 2002

1

Concepts

This chapter contains an introduction to the concepts behind Application Developer and an overview of the features of the various members of the WebSphere Studio family of tools. The following topics will be discussed:

- Eclipse
- Workbench architecture
- Workbench features
- WebSphere Studio products
- Tools
- WebSphere Studio and VisualAge for Java
- The sample code in this book

© Copyright IBM Corp. 2002

1.1 Introduction

WebSphere Studio Application Developer (also known as Application Developer) is a one of the WebSphere Studio family of products that has been developed based on the Eclipse Workbench.

The *Eclipse Workbench* is an open source platform, designed by IBM and released to the open source community. It is an open, portable, universal tooling platform that provides frameworks, services and tools for building tools.

In essence the workbench provides the tool infrastructure. With this infrastructure in place, the tool builders are able to focus on the actual building of their tools. The workbench has been designed for maximum flexibility to support the development of tools for new technologies that may emerge in the future.

Development tools written for the workbench should support a *role-based development model,* in which the outcomes of the developers' work will be consistent. The developers should not have to be concerned with how different individual tools may be treating their files.

The WebSphere Studio family of products is an integrated platform (IDE) for development, testing, debugging and deploying. It provides support for each phase of the application development life cycle.

1.2 Eclipse

The *eclipse.org Consortium* (see logo Figure 1-1) was formed to deliver a new generation of application development tools.


eclipse
THE ECLIPSE PROJECT
(c) Copyright IBM Corp. 2000, 2001. All rights reserved. Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S., other countries, or both.

Figure 1-1 Eclipse logo

Industry leaders Borland, IBM, Merant, QNX Software Systems, Rational Software, RedHat, SuSE, TogetherSoft, and WebGain formed the initial eclipse.org board of directors and began work on the Eclipse open source project. See this URL http://www.eclipse.org for more details about this project.

All of the participating companies plan to release Eclipse Platform compatible product offerings.

In the Eclipse Platform, code access and use is controlled through the *Common Public License*, which allows individuals to create derivative works that are royalty free and have worldwide re-distribution rights.

The Eclipse Workbench is becoming a very attractive platform for tool developers. Independent Software Vendors (ISVs) are able to use the same APIs as IBM to create tools that will run on the Eclipse Workbench.

The Eclipse Workbench allows for a loose or tight integration of tooling with the workbench and with other tools in the workbench.

A simple tool can be a single plug-in, while more complex tools may consist of many separate plug-ins. The Eclipse component architecture allows ISVs to integrate as they scale from small to large tools.

Another reason why the Eclipse Workbench is becoming attractive to ISVs is that it offers the infrastructure and integration points allowing all tools to be integrated into the workbench, as well as a seamless integration between the tools themselves.

ISVs can now integrate their tool into the environment using the same user interface (UI) as all the other tools. That means that the way different objects are represented and maintained in the workbench is consistent between different tools. This allows one tool to work with an object in the same way as another tool.

Another key integration point is the way objects are stored in the workbench. Objects created and used by the tools are stored in the file system or in a repository using a common interface. This permits different tools to access objects in a consistent manner.

1.3 Workbench architecture

The basic architecture of the Eclipse Workbench is shown in Figure 1-2.

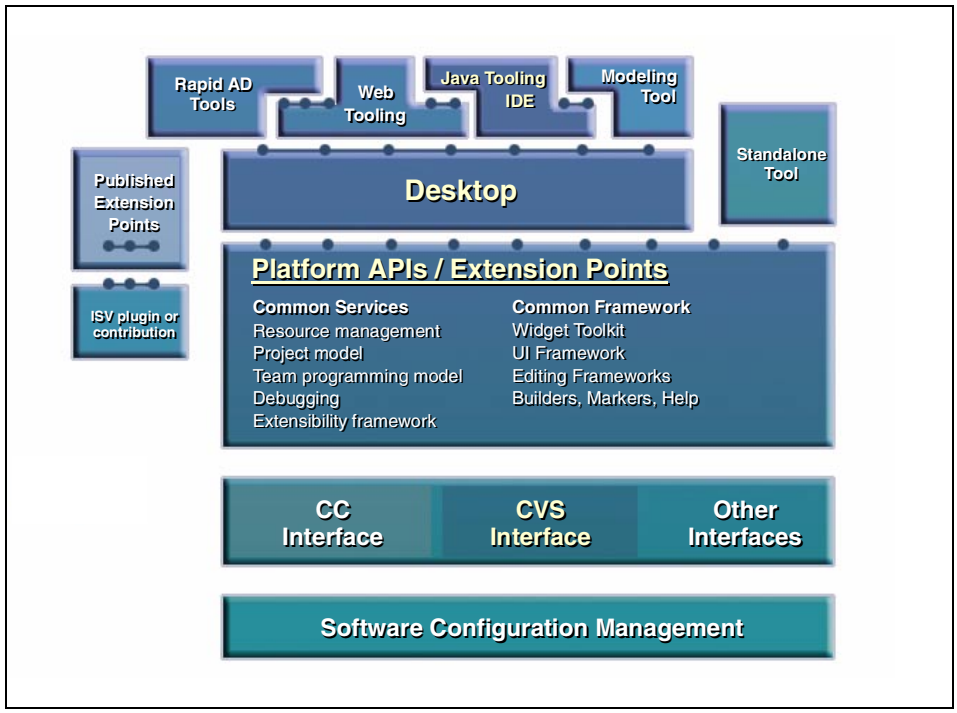


Figure 1-2 Workbench architecture

The workbench provides frameworks, services and tools that enable tool builders to focus on tool building, not on building the tool infrastructure:

- Editor frameworks
- UI frameworks
- SCM Adapters and team programming models
- Resource and project management
- Globalization
- Debugging

The platform core provides the integration points (called *extension points*) for plug-ins. These are used to add tools to the platform by implementing one or more of the extension points.

As an example, one set of extension points is for version and configuration management. In Application Developer this set is currently implemented by Concurrent Version System (CVS) and Rational ClearCase LT.

1.4 Workbench features

The Eclipse Workbench provides a set of APIs, models and frameworks for developing source editors and other user interfaces, as well as access to common services for resource management, debugging, and team programming.

The main features of the Eclipse Workbench are:

- Plug-in based tooling
- Role based development tools
- Vertical and horizontal integration
- Open standards
- Open team environment
- File based IDE

1.4.1 Plug-in based tooling

The workbench provides the ultimate plug-in platform. Different tools can plug in to the workbench, with each tool providing new functionality to be added to the workbench or to already existing plug-ins. Each plug-in integrates with the workbench and with the other tools. Ideally the end-user should not notice any difference when moving from one tool to another.

Eclipse has been designed from the ground up as a base for building integrated Web and application development tooling. The value of the platform is the rapid development of integrated features based on a plug-in model.

By programming to the portable Eclipse APIs, plug-ins can run unchanged on any of the supported operating systems using a common user interface model. The workbench is designed to run on multiple operating systems while providing robust integration with each one.

Each plug-in can focus on performing a small number of tasks well, without having to provide the supporting infrastructure. Some examples of such tasks are defining, testing, animating, publishing, compiling, debugging and diagramming.

Since the workbench is based on an open architecture, each plug-in development team can focus on their area of expertise. This enables the team management experts to build the back end interfaces and the usability experts to build the end user tools. If these are well designed, and use the standard APIs, significant new features and levels of integration can be added to the workbench and workbench base products without impacting other tools.

The workbench manages the complexity of different runtime environments, such as different operating systems and workgroup server environments. Plug-in developers can focus on their specific tasks instead of worrying about such integration issues.

1.4.2 Role-based development with consistent look and feel

The workbench is designed to provide special support for a particular e-business development role, or for a range of roles.

Within the workbench based products, task-oriented perspectives filter out much of the overall Web and Java development complexity, and present the developer only with those functions that are relevant to the task at hand.

Users can switch perspectives depending on what they are working on at any given moment, or depending on their current role in the project.

Because different developers are accustomed to working in different ways, any perspective can be further customized. Since they are built using the Eclipse Workbench technology, all tools and perspectives share a common look and feel, which reduces learning curves and help maximize developer productivity.

Since all development resources for all projects are stored in a single repository, developers have consistent team support for their projects, and are able to easily share their work products.

1.4.3 Vertical and horizontal integration

Traditionally software vendors have provided vertical tools, forcing customers to do their own integration. The purpose of the Eclipse Workbench is to provide a platform that software vendors can easily extend. ISVs have already embraced this technology and are actively building tools on this base.

As an example, every WebSphere Studio family of products that is built on the Workbench offers a set of already integrated tools, freeing you to focus on building applications rather than on integrating tools. Furthermore you can easily integrate other tools, (from other vendors or locally developed), as long as they conform to the workbench standard plug-in protocol.

1.4.4 Open standards

The whole Eclipse Workbench, as well as all products of the WebSphere Studio family of products, are built on open standards and the code that they generate also complies with open standards.

This allows you to build and deploy state-of-the-art, server-side applications that conform to the Servlets 2.2, JavaServer Pages 1.1, and EJB 1.1 specifications.

1.4.5 Open team development

Application development teams are becoming more distributed, more diverse, and are under increasing pressure to deliver solutions quickly. In such an environment it is critical to have a development environment that can support these needs, while at the same time addressing personalized requirements. The team development environment for all products based on the Eclipse Workbench supports pluggable repositories rather than mandating any proprietary repository, and support an optimistic concurrency model.

1.4.6 File-based IDE

The Eclipse Workbench is a platform for building file-based IDEs. All content is saved as files. Workbench resources, such as Java classes and HTML files, are stored in the file system, making them easy to access.

1.5 WebSphere Studio product family

The WebSphere Studio family of products is built on the top of the Eclipse Workbench as a set of plug-ins conforming to the workbench's open standard APIs. These products are then follow-on technology for WebSphere Studio Advanced Edition V3 and VisualAge for Java Enterprise Edition V4.

The WebSphere Studio family of products currently has the following members Figure 1-3:

- WebSphere Studio Site Developer Advanced
- WebSphere Studio Application Developer
- WebSphere Studio Application Developer Integration Edition
- WebSphere Studio Enterprise Developer

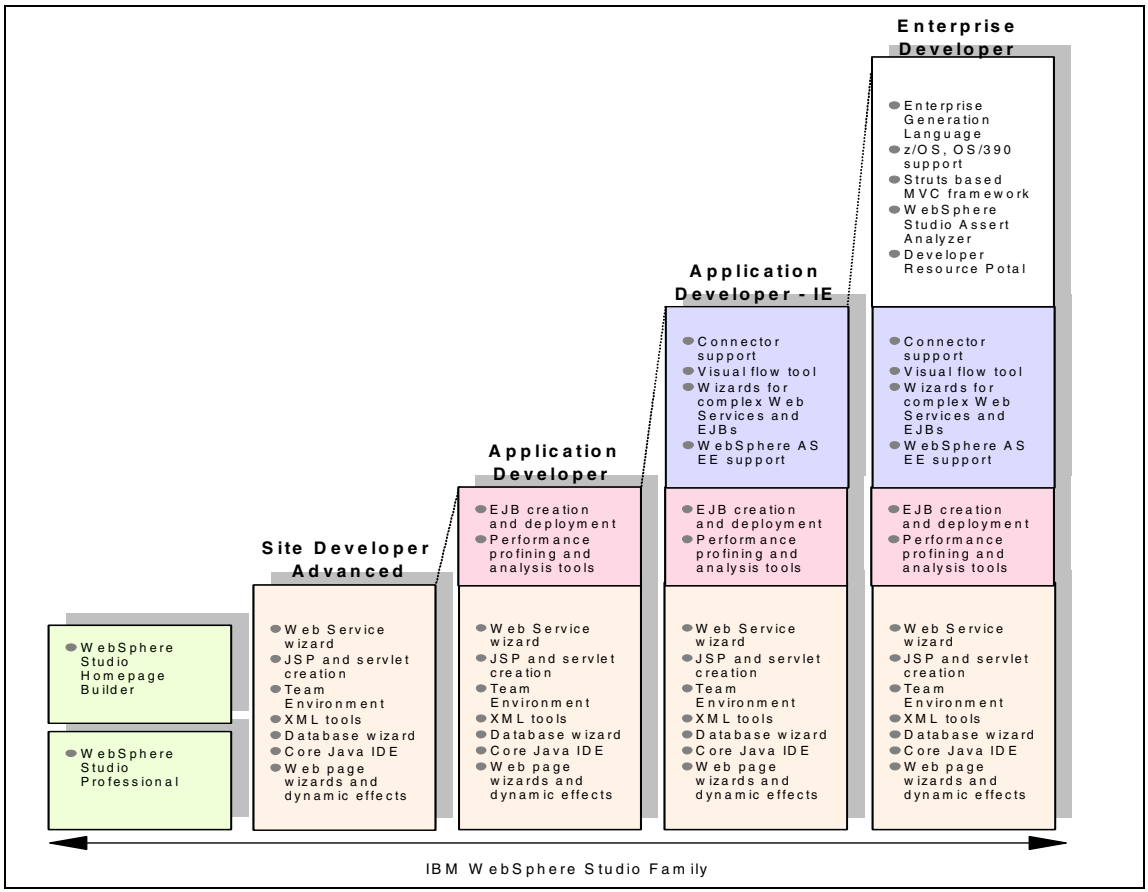


Figure 1-3 WebSphere Studio family of products

These products provide support for end-to-end development, testing, and deployment of Web and J2EE applications.

The WebSphere Studio family of products provide integrated development tools for all e-business development roles including Web developers, Java developers, business analysts, architects, and enterprise programmers. The customizable, targeted and role-based approach of the workbench will be a common characteristic of future products in the WebSphere Studio family.

WebSphere Studio Site Developer Advanced

Site Developer Advanced is an IDE intended for Web developers who develop and manage complex Web sites. It is an easy-to-use toolset that minimizes the time and effort required to create, manage, and debug multi platform Web sites. It is designed according to the J2SE and J2EE specifications and supports JSPs, servlets, HTML, JavaScript, and DHTML. It further includes tools for developing images and animated GIFs.

Site Developer Advanced enables Web developers to use their favorite content creation tools in conjunction with the built-in local and remote publishing capabilities.

Using Site Developer Advanced, you can develop Web applications that use the following technologies.

- **JSPs:** A simple, fast and consistent way to extend Web server functionality and create dynamic Web content. JSPs enable rapid development of Web applications that are server and platform-independent.
- **Servlets:** Server code that executes within a Web Application Server.
- **Web services:** Self-contained, modular applications that can be described, published, located, and invoked over the Internet or within intranets.

WebSphere Studio Application Developer

Application Developer is designed for professional developers of Java and J2EE applications, who require integrated Web, XML, and Web services support.

It includes all of the features of Site Developer Advanced, and adds tools for developing EJB applications, as well as performance profiling and logging tools for both local and remote execution.

Developers can quickly build and test business logic and enhance the presentation artifacts with built-in Web creation tools inside the Application Developer IDE before deploying to a production server.

Using the performance profiling and tracing tools makes it possible to detect application performance bottlenecks earlier in the development cycle. Furthermore, the built-in test environment for WebSphere Application Server and advanced tools for code generation help to shorten the test cycle.

WebSphere Studio Application Developer Integration Edition

Integration Edition includes all of the functionality in Application Developer, plus:

- Powerful graphical tools to help you quickly and easily build custom application adapters to integrate your J2EE application with your back-end systems, helping you save time and money by reusing existing resources.

- Visual flow-based tools that increase developer productivity by allowing them to visually define the sequence and flow of information between application artifacts such as adapters, Enterprise JavaBeans components and Web services.
- Wizards that help in building and deploying complex Web services out of adapters, EJB components, flows, and other Web services.
- Support for the full set of Enterprise services provided by WebSphere Application Server Enterprise Edition such as Business Rule Beans, internationalization, and work areas that deliver additional integration capabilities, developer productivity, and business agility.

WebSphere Enterprise Developer

Enterprise Developer includes all of the functionality in WebSphere Studio Application Developer Integration Edition, plus:

Enterprise Developer can be used to implement Struts-based MVC applications using connectors and the Enterprise Generation Language (EGL).

The ability to connect components is the first step in modernizing the application portfolio of enterprises. It supports creating and connecting Web applications to Enterprise business logic using the Struts-based Model-View-Controller framework and associated tooling.

Two other core technologies are integrated within Enterprise Developer.

- **WebSphere Studio Asset Analyzer (WSAA)**: Identifies application processes and connecting points, and provides the ability to generate components from existing code.
- **Developer Resource Portal (DRP):** Provides collaborative capabilities across the entire development process.

Enterprise Developer addresses the needs of large enterprises, providing a model based paradigm for building applications in a Struts-based Model-View-Controller framework. It provides a visual construction and assembly based environment supporting the implementation of enterprise level applications, including support for the multiple developer roles and technologies required by those applications. Some examples of technologies supported are HTML, JSPs, servlets, EJBs, COBOL, EGL, PL/I and connectors.

1.6 Tools

The WebSphere Studio family of products include the following basic tools:

- Web development tools
- Relational database tools
- XML tools
- Java development tools
- Web services development tools
- Team collaboration tools
- Integrated debugger
- Server tools for testing and deployment
- Enterprise JavaBean development tools (not in Site Developer Advanced)
- Performance profiling tools (not in Site Developer Advanced)
- Plug-in development tools

1.6.1 Web development tools

The professional Web development environment provides the necessary tools to develop sophisticated Web applications consisting of static HTML pages, JSPs, servlets, XML deployment descriptors and other resources.

Wizards are available to generate running Web applications based on SQL queries and JavaBeans. Links between Web pages can be automatically updated when content changes. There are also tools for creating images and animated GIFs.

The Web development environment bring all aspects of Web application development into one common interface. Everyone on your Web development team including content authors, graphic artists, programmers and Web masters, can work on the same projects and access the files they need.

Such an integrated Web development environment makes it easy to collaboratively create, assemble, publish, deploy, and maintain dynamic, interactive Web applications.

The Web development tools provide the following features:

- Support for latest Web technology with an intuitive user interface.
- Advanced scripting support to create client-side dynamic applications with VBScript or JavaScript.
- Web Art Designer to create graphic titles, logos, buttons, and photo frames with professional-looking touches.
- Animated GIF Designer to create life-like animation from still pictures, graphics, and animated banners.

- Over 2,000 images and sounds in the built-in library.
- Integrated, easy-to-use visual layout tool for JSP and HTML file creation and editing.
- Web project creation, using the J2EE-defined hierarchy.
- Creation and visual editing of the Web application deployment descriptor (web.xml) file.
- Automatic update of links as resources are moved or renamed.
- A wizard for creating servlets.
- Generation of Web applications from database queries and JavaBeans.
- J2EE WAR/EAR deployment support (not in Site Developer Advanced).
- Integration with the WebSphere unit test environment.

1.6.2 Relational database tools

The database tools provided with the WebSphere family products allow you to create and manipulate the data design for your project in terms of relational database schemas.

You can explore, import, design, and query databases working with a local copy of an already existing design. You can also create an entirely new data design from scratch to meet your requirements.

The database tools provide a metadata model that is used by all other tools that need relational database information. This includes database connection information. In that way tools, although unaware of each other, are able to share connections.

The SQL Statement wizard and SQL Query Builder provide a GUI-based interface for creating and executing SQL statements. When you are satisfied with your statement you can use the SQL to XML wizard to create an XML document, as well as XSL, DTD, XSD, HTML and other related artifacts.

The relational database tools support connecting to, and importing from, several database types, including DB2, Oracle, SQL Server, Sybase, and Informix.

1.6.3 XML tools

The comprehensive XML toolset provided by the WebSphere Studio family of products includes components for building DTDs, XML schemas and XML files. With the XML tools you can perform all of the following tasks:

- Create, view, and validate DTDs, XML schemas, and XML files.
- Create XML documents from a DTD, from an XML schema, or from scratch.
- Generate JavaBeans from a DTD or XML schema.
- Define mappings between XML documents and generate XSLT scripts that transform documents.
- Create an HTML or XML document by applying an XSL style sheet to an XML document.
- Map XML files to create an XSL transformation script and to visually step through the XSL file.
- Define mappings between relational tables and DTD files, or between SQL statements and DTD files, to generate a document access definition (DAD) script, used by IBM DB2 XML Extender. This can be used to either compose XML documents from existing DB2 data or decompose XML documents into DB2 data.
- Generate DADX, XML, and related artifacts from SQL statements and use these files to implement your query in other applications.

1.6.4 Java development tools

All WebSphere Studio family of products provide a professional-grade Java development environment with the following capabilities:

- Application Developer v4.0 ships with JDK 1.3
- Pluggable run-time support for JRE switching and targeting of multiple run-time environments from IBM and other vendors.
- Incremental compilation.
- Ability to run code with errors in methods.
- Crash protection and auto-recovery.
- Error reporting and correction.
- Java text editor with full syntax highlighting and complete content assist.
- Refactoring tools for reorganizing Java applications.
- Intelligent search, compare, and merge tools for Java source files.
- Scrapbook for evaluating code snippets.

1.6.5 Web services development tools

Web services represent the next level of function and efficiency in e-business. Web services are modular, standards-based e-business applications that businesses can dynamically mix and match in order to perform complex transactions with minimal programming. The WebSphere Studio family of products that include the Web services feature, help you to build and deploy Web services-enabled applications across the broadest range of software and hardware platforms used by today's businesses. These tools are based on open, cross-platform standards such as Universal Description Discovery and Integration (UDDI), Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL).

1.6.6 EJB development tools

The WebSphere Studio family of products, except Site Developer Advanced, feature full EJB support (Application Developer v4.0 supports EJB 1.1), an updated EJB test client, an enhanced unit test environment for J2EE, and deployment support for Web application archive (WAR) files and enterprise application archive (EAR) files. Entity beans can be mapped to databases, and EJB components can be generated to tie into transaction processing systems. XML provides an extended format for deployment descriptors within EJB.

1.6.7 Team collaboration

Team developers do all of their work in their individual workbenches, and then periodically make changes to a "team stream." This model allows individual developers to work on a team project, share their work with others as changes are made, and access the work of other developers as the project evolves. At any time, developers can update their workspaces by retrieving the changes that have been made to the team stream or by submitting changes to the team stream.

All products of the WebSphere Studio family support the Concurrent Versions System (CVS) and the Rational ClearCase LT products among others.

Other software configuration management (SCM) repositories can be integrated through the Eclipse Workbench SCM adapters. SCM adapters for commercial products are provided by the vendors of those products.

1.6.8 Debugging tools

The WebSphere Studio family of products include a debugger that enables you to detect and diagnose errors in your programs running either locally or remotely. The debugger allows you to control the execution of your program by setting breakpoints, suspending launches, stepping through your code, and examining the contents of variables.

You can debug live server-side code as well as programs running locally on your workstation.

The debugger includes a debug view that shows threads and stack frames, a process view that shows all currently running and recently terminated processes, and a console view that allows developers to interact with running processes.

There are also views that display breakpoints and allow you to inspect variables.

1.6.9 Performance profiling tools

The WebSphere Studio family of products, except Site Developer Advanced, provide tools that enable you to test the performance of your application. This allows you to make architectural and implementation changes early in your development cycle, and significantly reduces the risk of finding serious problems in the final performance tests.

The profiling tools collect data related to a Java program's run-time behavior, and present this data in graphical and non-graphical views. This assists you in visualizing program execution and exploring different patterns within the program.

These tools are useful for performance analysis and for gaining a deeper understanding of your Java programs. You can view object creation and garbage collection, execution sequences, thread interaction, and object references. The tools also shows you which operations take the most time, and help you find and plug memory leaks. You can easily identify repetitive execution behavior and eliminate redundancy, while focusing on the highlights of the execution.

1.6.10 Server tools for testing and deployment

The server tools provide a unit test environment where you can test JSPs, servlets and HTML files, (EJB testing is supported in Application Developer and Enterprise Developer). You also have the capability to configure other local or remote servers for integrated testing and debugging of J2EE applications.

The following features are included:

- A copy of the complete WebSphere Application Server Developer Edition (AEs) run-time environment.
- Standalone unit testing.
- Ability to debug live server-side code using the integrated debugger.
- Support for configuring multiple servers.

The server tools support the following run-time environments:

- WebSphere Application Server AEs Version 4.01, which can be installed locally or remotely. It supports testing of both EJBs and Web applications.
- Apache Tomcat, which can be installed only locally and supports testing of Web applications.

1.6.11 Plug-in development tools

The WebSphere Studio family of products, except Site Developer Advanced, include the plug-in development environment (PDE) that is designed to help you develop platform plug-ins while working inside the platform workbench and it provides a set of platform extension contributions (views, editors, perspectives, etc.) that collectively streamline the process of developing plug-ins inside the workbench. The PDE is not a separate tool but it is a one of perspectives. PDE blends with the platform and offers its capabilities through a new perspective.

The following project types are supported:

- **Plug-in project**: WebSphere Studio Application Developer is based on the concept of plug-ins that have a clearly defined structure and specification. This project supports to create, test, and deploy a plug-in in the PDE.
- **Fragment project**: A plug-in fragment is used to provide additional plug-in functionality to an existing plug-in after it has been installed. Fragments are ideal for shipping features like language or maintenance packs that typically trail the initial products by a few months.
- **Plug-in component**: PDE attaches a special **component** nature to plug-in and fragment projects to differentiate them from other project types. The project must have a specific folder structure and a component manifest. The project must be set up with references to all of the plug-in and fragment projects that will be packaged into the component.

1.7 WebSphere Studio V4 and VisualAge for Java

The WebSphere Studio family of products is the follow-on technology for WebSphere Studio Advanced Edition V4 and VisualAge for Java Enterprise Edition V4.

From the perspective of current VisualAge for Java users, the WebSphere Studio family of products adds many new features, including HTML and JSP editors, rich media tools, XML development tools, and Web services wizards.

In terms of Java development, Application Developer focuses on J2EE server-side (EJB and servlet) development, deployment, and profiling.

VisualAge for Java still provides a wider scope, including client application development with Swing using the Visual Composition Editor, and tools like Enterprise Access Builder for accessing back-end transaction servers that facilitate working with many kinds of legacy systems.

However most of the features from WebSphere Studio and VisualAge for Java will eventually be integrated into the new WebSphere Studio family of products.

For instructions and tips on migrating your applications from VisualAge for Java, see Chapter 14, “Migrating your VisualAge for Java project” on page 339.

1.8 Sample code in this book

The sample code introduced in this redbook (Figure 1-4) is based on a simple application scenario, involving a car parts inventory.

Each sample accesses a DB2 table that contains the car parts and displays the result to the user. The Servlet/JSP/JavaBean sample is designed to demonstrate the use of the Model-View-Controller (MVC) model for developing Web applications.

The following sample code is created in this book:

- **Query Application**: (`Listing.java`) - A stand alone Java application. Gets a search key from the request argument and returns the result to the console (Chapter 6, “Creating Java applications” on page 139).
- **Query Applet**: (`PartListApplet.java`) - A Java Applet. Works within a Web page. GUI version of Listing.java. TestApplet.html is provided to test the applet (Chapter 6, “Creating Java applications” on page 139).
- **Title Web page**: (`index.html`) - A simple, static HTML Web page (Chapter 7, “Creating HTML resources” on page 161).

- **Query input form**: (`PartList.html`) - Part list inquiry entry form (Chapter 7, “Creating HTML resources” on page 161).
- **MVC** sample (Chapter 8, “Creating Web applications with dynamic content” on page 203):
  - **Controller Servlet**: (`PartListServlet.java`) - Part list servlet. Works as a controller in the MVC model.
  - **Business Logic**: (`PartListBean.java`) - JavaBean. Works as a model in the MVC model. It contains the business logic to connect to the database and get the result.
  - **Query Result**: (`PartList.jsp`) - A JavaServer Page. Works as a view in the MVC model. This JSP shows the result from the inquiry using PartListBean.
- **Stored Procedure**: (`PartListing.java`) A DB2 stored procedure to read the parts table. PartListBeanSP.java is a bean using the stored procedure to retrieve and display the result Chapter 11, “Stored procedures” on page 297.

Note: This redbook does not in detail cover EJB, XML or Web Services development using Application Developer. However, an introduction to the tools provided with Application Developer to support these features is included in Chapter 3, “Perspectives, views, and editors” on page 33.

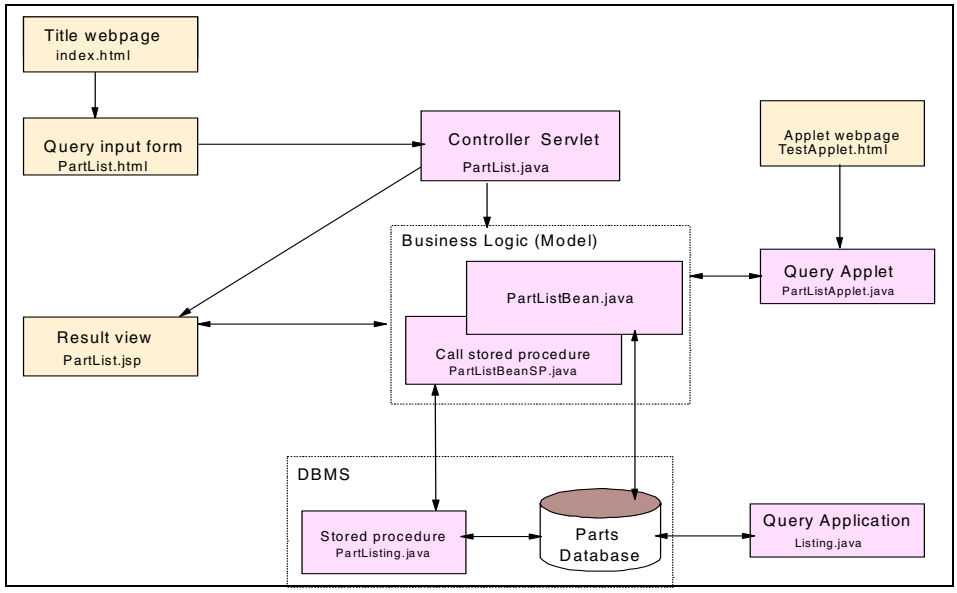


Figure 1-4 Redbook sample code

1.9 Sample plug-ins in this book

The following sample plug-ins are created in this book:

- `ITSOJarViewer`

  This plug-in has a window in the workbench and shows the content of jar file using views extension point.

- `ITSOJarEditor`

  This plug-in is similar to ITSOJarViewer but it uses editor extension point.

- `ITSOProjectClasspath`

  This plug-in is Ant Task plugin which is using JDT APIs.

2

Setting up your workbench

After you have installed Application Developer, and before you start creating your projects, you may want to modify some of the default workbench settings to suit your needs or site standards. This chapter will describe the most important areas where you can customize the Application Developer setup.

The following topics will be covered in this chapter:

- Java class path variables
- Java coding preferences
- Automatic builds
- Workbench window preferences

© Copyright IBM Corp. 2002

2.1 Defining Java class path variables

An important task when setting up your workbench for Java coding is to define additional class path variables. Application Developer will automatically create a number of default class path variables. Depending on the type of Java coding you plan to do, and any legacy code that you want to make use of, you may need to add variables pointing to other code libraries. Some common examples are:

- The DB2 JDBC driver class
- VisualAge for Java Libraries (for example Data Access Bean classes)
- Locally developed common code

After you have created a new Java project, you can add any of these variables to the project's class path. How to set up the class path for a project is discussed in detail in "Java applications" on page 140.

To add a new Java class path variable, select:
Window—>Preferences—>Java—>Classpath Variables. A list of existing class path variables will be displayed Figure 2-1.

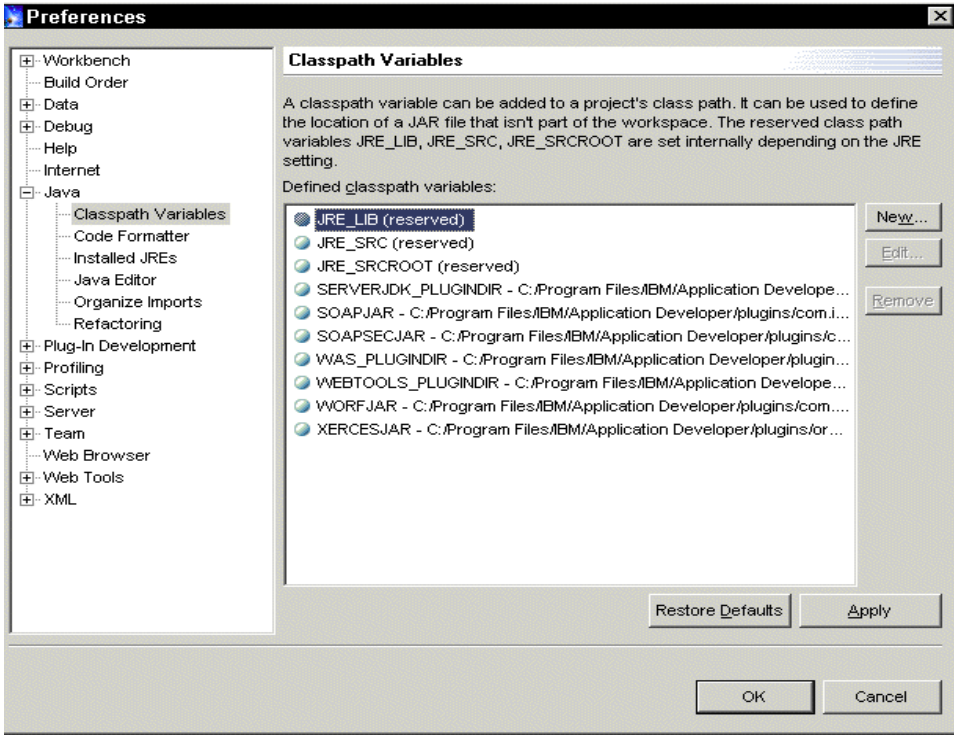


Figure 2-1 Defined class path variables

In this dialog you can edit existing variables, add new ones and remove ones that are no longer required.

Click **New** to add a variable. In the example shown in Figure 2-2, we are adding a variable pointing to the DB2 JDBC driver class. The name of the file containing the class, `db2java.zip`, is specified along with the file system path to it.

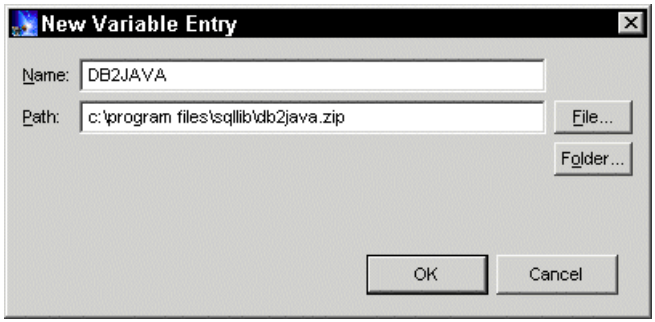


Figure 2-2 Adding a Java class path variable

2.2 Java coding preferences

There are a number of Java coding preferences that you can modify to suit your needs:

- Code formatting
- Java editor preferences
- Import organization
- Refactoring
- Choice of JRE

2.2.1 Code formatting

The Java editor in the workbench can be configured to format code in conformance with personal preferences or team standards. When setting up the workbench you can decide what formatting should be applied. To modify the default code formatting select: **Windows—>Preferences—>Java—>Code Formatter** (Figure 2-3). Use the tabs at the top of the page to modify various aspects of the code formatting. The code sample in the bottom right pane will show you a preview of the effects of changes that you make.

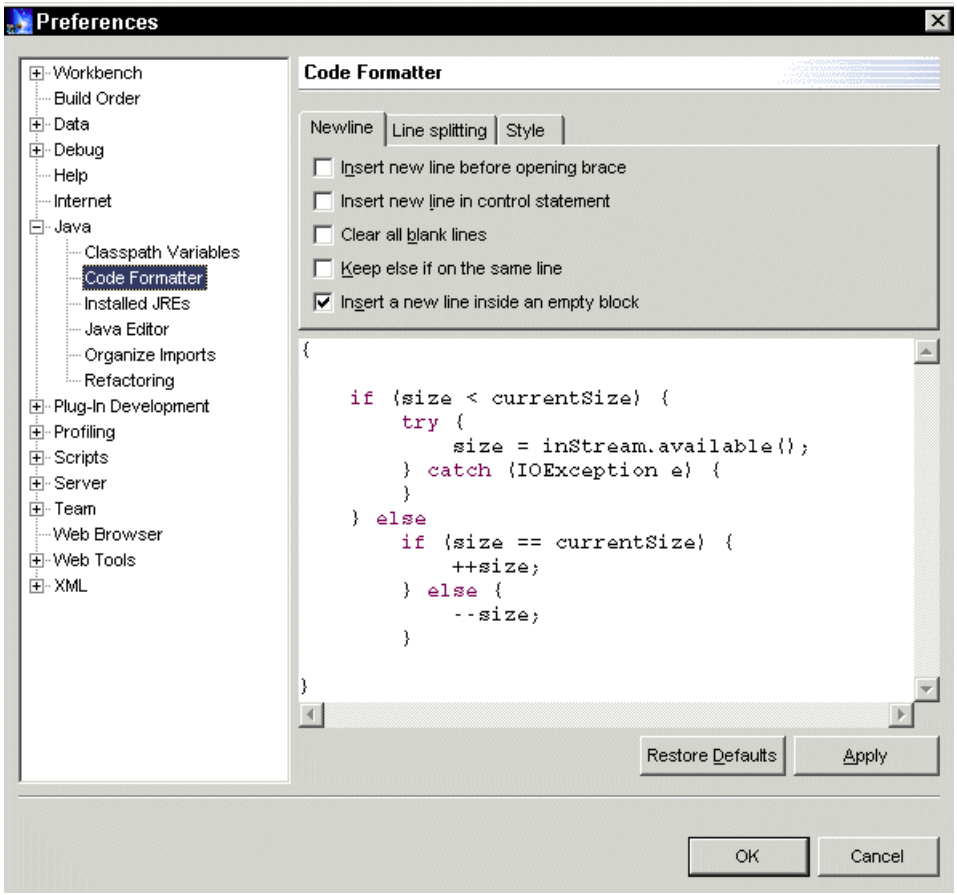


Figure 2-3 Code formatting options

Note: To apply the formatting rules defined here, select **Format** from the Java editor context menu. (Some formatting, for example indenting of braces, will be done on the fly while you are editing the source code.)

2.2.2 Java editor preferences

By default the Java editor will use the standard workbench font. If you want, you can change the font used in the editor. To choose another font to use for all Java editor windows, select **Windows—>Preferences—>Java—>Java Editor** (Figure 2-4). Select **Change** to see a list of available fonts.

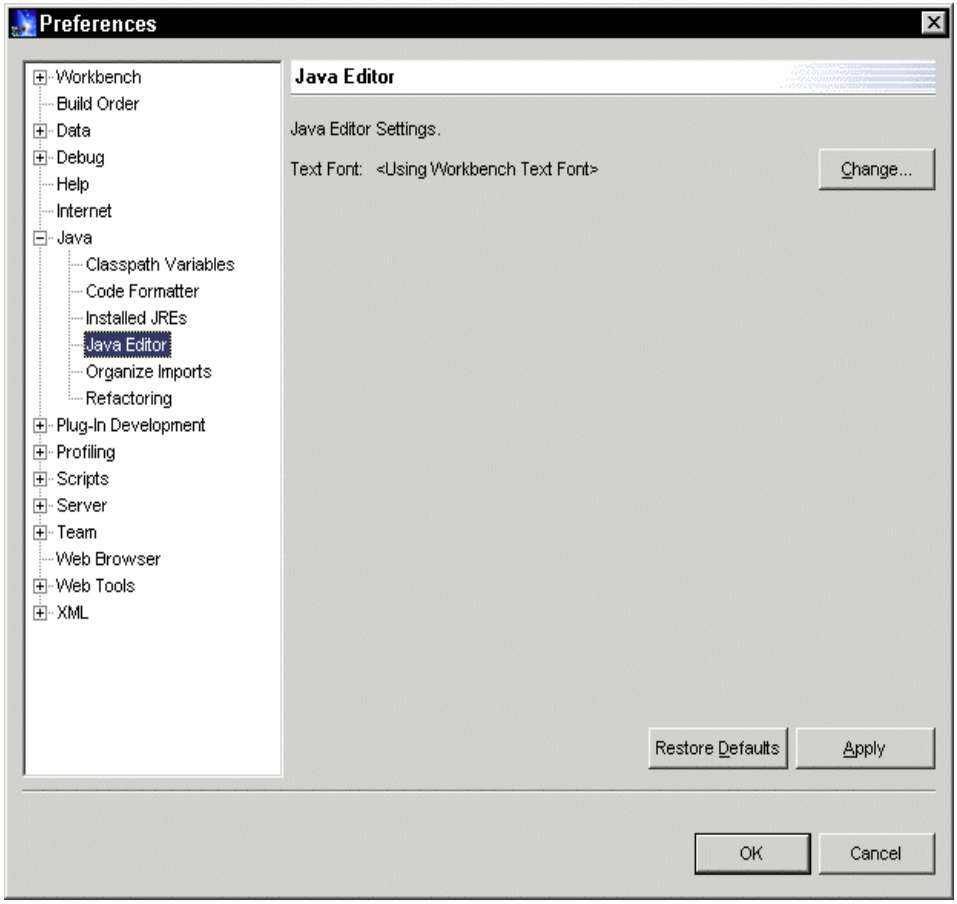


Figure 2-4 Java editor preferences

2.2.3 Import organization

You can specify how you want the Java editor to handle imports when using the automatic import generation feature. See “Import generation” on page 125 for a description of this feature. To display the Preference dialog select **Windows—>Preferences—>Java—>Organize Imports** (Figure 2-5).

In the preferences you can specify the order of the import statements. You can also control at what stage `<package name>.*` import statements should be generated rather than fully qualified import statements. The default for when to switch to the `<package name>.*` syntax is after 99 qualified imports from the same package.

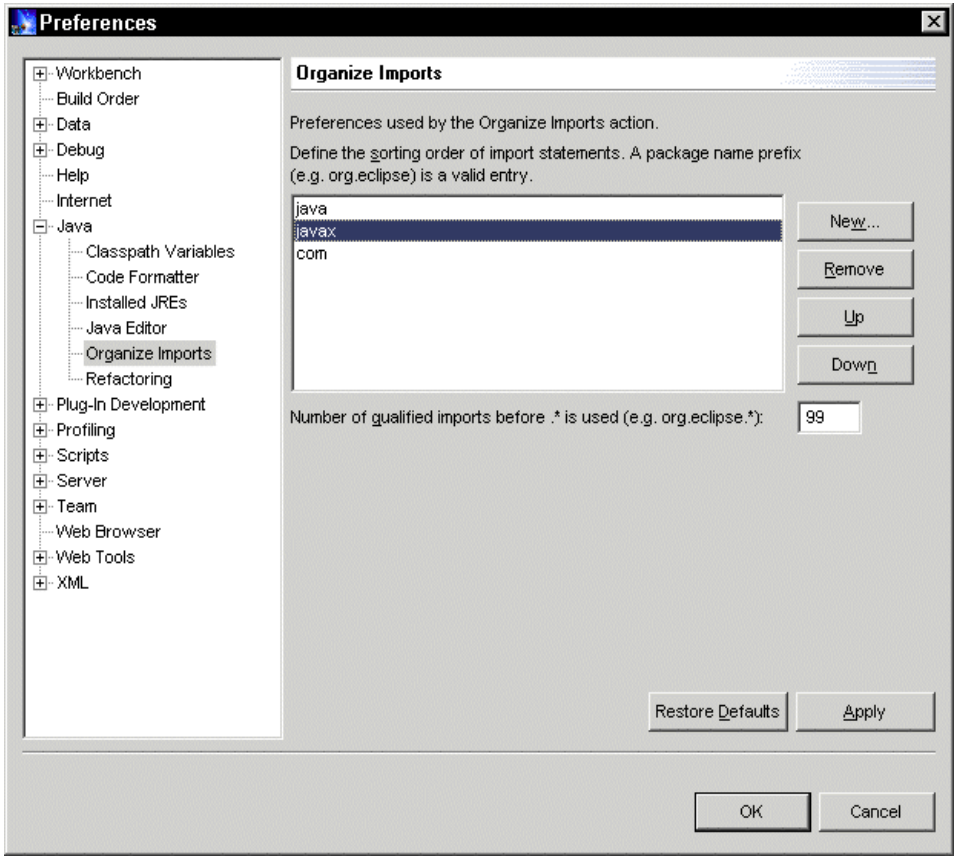


Figure 2-5 Organize imports

Use the **Up** and **Down** buttons to change the order of imports. You can also add a new import to the list and place it in the import order.

2.2.4 Refactoring

Refactoring refers to the process of moving or renaming Java elements. In the workbench setup you can set preferences as to how this process will work. See “Refactoring” on page 127 for a detailed discussion about refactoring and how the preferences you select during workbench setup affect how the process works.

2.2.5 Choice of JRE

Application Developer allows you to specify which JRE should be used by the Java builder. By default the standard Java VM that comes with Application Developer is used, but if you have special requirements you may want to add another JRE to be used as default or for special projects. You select **Windows—>Preferences—>Java—>Installed JREs** to display the dialog (Figure 2-6).

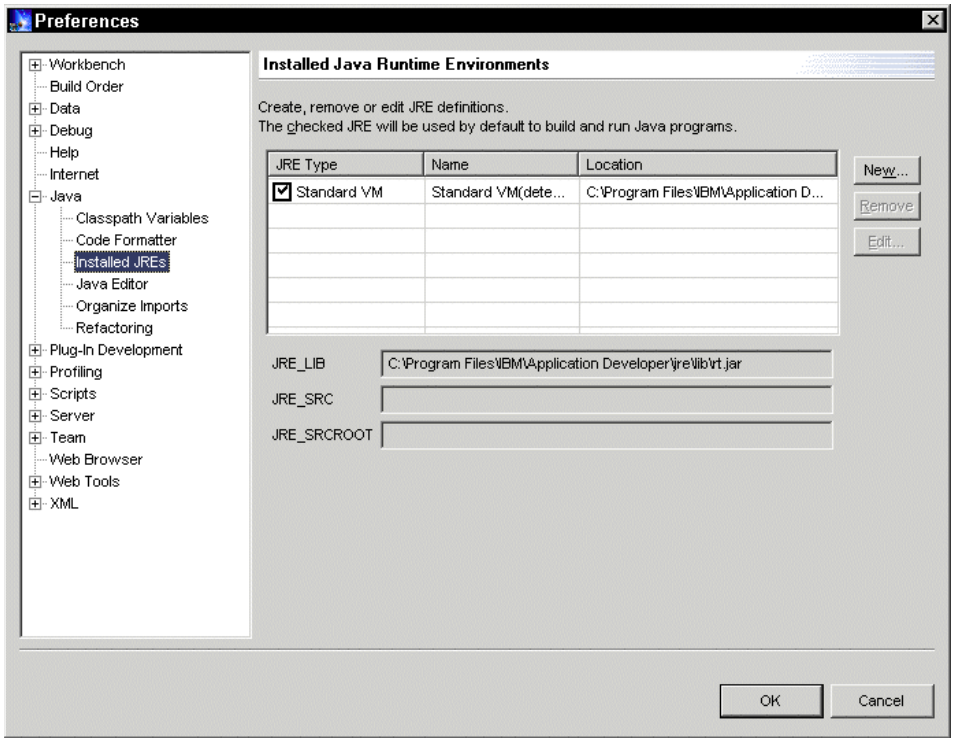


Figure 2-6 Installed JREs

The current default JRE is indicated by the check box to the left of the JRE type. You can use this dialog to add another JRE and to indicate whether this should be the default JRE or not. When setting up a project you can choose which of the JREs to use. If you don't explicitly specify the JRE for the project, the default one will be used.

To add a new JRE click the **New** button. The following dialog will be displayed (Figure 2-7).

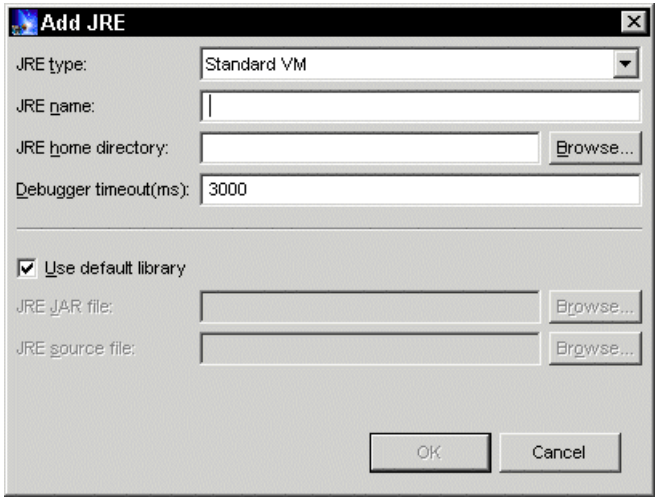


Figure 2-7 Add JRE

Select the type of JRE, give it a unique name within that type, and specify the directory where it resides on the file system. The system will attempt to determine the library paths to use. If this is successful the fields **JRE JAR file** and **JRE source file** will be filled in. If not, or if you want to change the defaults, uncheck the **Use default library** check box and enter or modify the values.

2.3 Automatic builds

By default, builds in Application Developer are done automatically whenever a resource has been modified. Under most circumstances this is what you want. However, if you require more control over when builds occur, you can temporarily disable the auto-building feature. To perform a build you will then have to explicitly start it. This may be desirable in cases where you know that building is of no value until you finish a large set of changes. In this case there is no benefit to incurring the overhead of auto-building.

Note: Builds in Application Developer are incremental, that is only resources that have been modified will be built. Project builds and rebuild all function will re-compile all instead of incremental compile.

If you want to turn off the automatic build feature select **Windows —> Preferences—>Workbench** and uncheck the **Perform build automatically on resource modification** check box.

In this dialog you can also specify whether or not you want unsaved resources to be saved before performing a manual build. Check the **Save all modified resources automatically prior to manual build** check box to enable this feature (Figure 2-8).

2.4 Workbench window preferences

The workbench consists of a number of perspectives and a number of windows specific to the perspective you are currently viewing. In the Workbench Preferences dialog you can specify how you want the workbench to behave when you open new perspectives within a project and when you create a new project. Select **Windows—>Preferences—>Workbench** to display the Workbench dialog (Figure 2-8). The default is to always open a perspective in the same window.

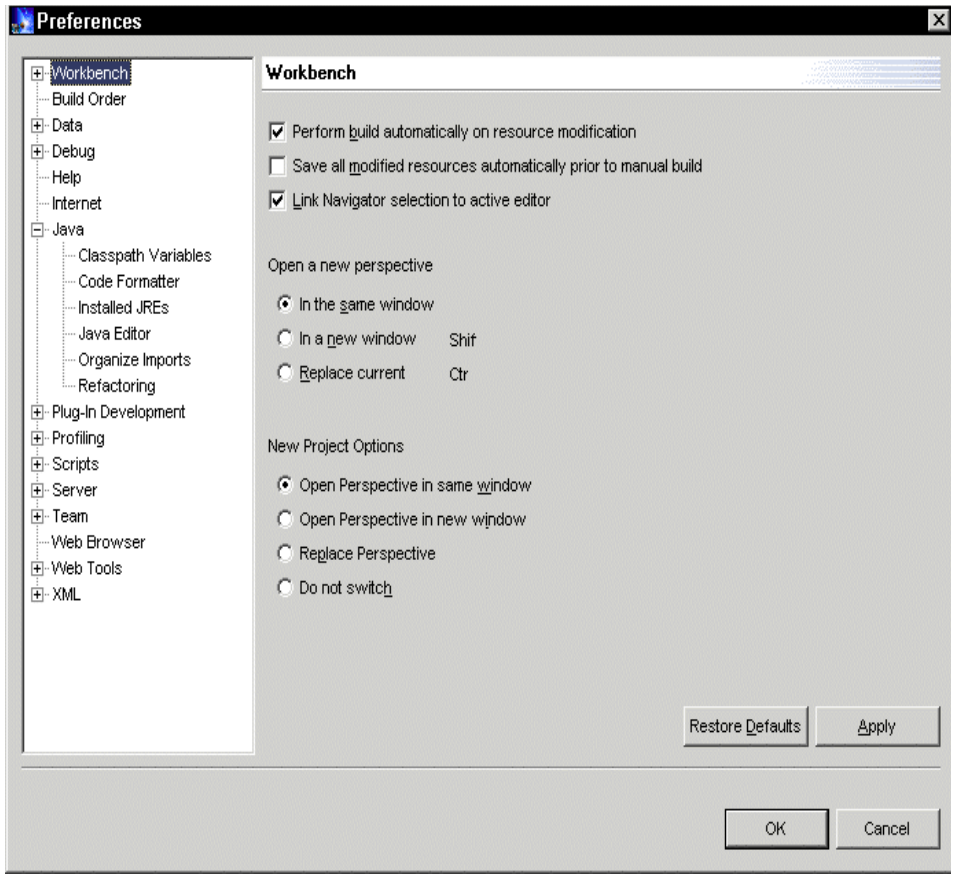


Figure 2-8 Workbench preferences

Note: When you open a new perspective you can always override the default behavior by pressing the Shift and Ctrl keys while performing the action. Holding down the Shift key will open the new perspective in a new window, and holding down the Ctrl key will replace the current perspective with the new one.

3

Perspectives, views, and editors

As already described in the “Introduction” on page 4, Application Developer supports a role based development model. It does so by providing several different perspectives on the same project. Each perspective is suited for a particular role and provides the developer with the necessary tools to work on the tasks associated with that role.

This chapter introduces the following topics:

- Resource perspective
- Java perspective
- Web perspective
- J2EE perspective
- Server perspective
- XML perspective
- Data perspective
- Debug perspective
- Profiling perspective
- Script perspective
- Team perspective
- Help perspective
- Customizable perspective
- Task list

© Copyright IBM Corp. 2002

3.1 Integrated Development Environment (IDE)

Application Developer contains a number of predefined perspectives and each perspective has several different views. This chapter introduces each perspective and view and we also briefly discuss the different default editors that are used to manipulate project resources. Application Developer also provides for special requirements by making it possible for developers to tailor the predefined perspectives to suit their individual or team needs.

Application Developer features an integrated development environment with customizable perspectives that support role-based development. It provides a common way for all members of a project team to create, manage, and navigate resources easily. It consists of a number of interrelated views and editors.

Views provide different ways of looking at the resources you are working on. Editors allow you to create and modify the resource. Perspectives are combinations of views and editors that show various aspects of the project resources, and are organized by developer role or task. For example, a Java developer would work most often in the Java perspective, while a Web designer would work in the Web perspective.

Several perspectives are provided; team members can customize them, according to their current role or preferences.

You can open more than one perspective at a time, and switch perspectives as you are working.

3.1.1 Perspectives

A perspective defines an initial set and layout of views and editors for performing a particular set of development activities. One or more perspectives can be open in a single workbench window.

You can switch between perspectives in the same window, or you can open them in new workbench windows. For example, you could keep the Help perspective open in a separate window as you work through a task.

You can create new perspectives or customize existing ones by adding, deleting, and rearranging views.

3.1.2 Views

Views provide alternative presentations of ways of navigating through the information in your workbench. For example, the Navigator view displays projects and other resources that you are working on within a folder hierarchy.

A view might appear by itself, or stacked with other views in a tabbed notebook arrangement.

A perspective determines the views you are likely to need. For example, the Java perspective includes the Packages view and the Hierarchy view to help you work with Java packages and hierarchies.

3.1.3 Editors

When you open a file, the workbench determines what type of file you have selected, based on the file extension, and opens the editor that is associated with that extension. File associations can be changed in the **Window —> Preferences—>Workbench—>File editors** dialog.

For example, the workbench opens an HTML editor for **.html**, **.htm** and **.jsp** files; a Java editor for **.java** and **.jav** files; and a text editor for **.txt** files.

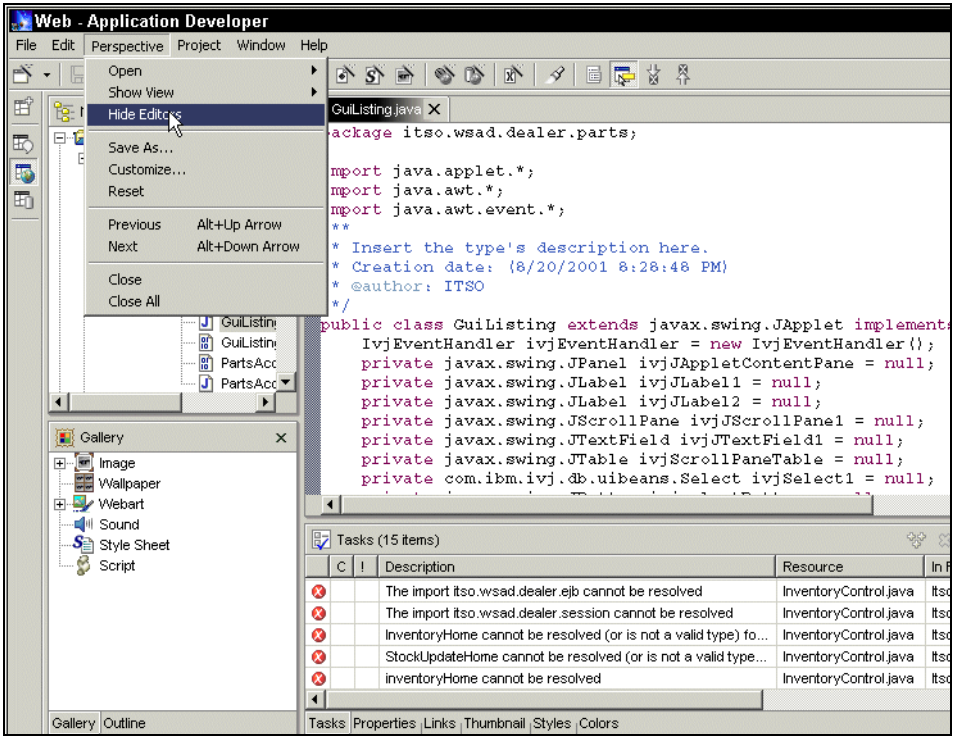


Figure 3-1 Showing/hiding the editor pane

You can show/hide the editors from your perspective by toggling the **Show Editors/Hide Editors** menu item in the **Perspective** menu (Figure 3-1), which can be found in the workbench menu bar.

You can add another view to your perspective by selecting the **Show View** menu item in the same **Perspective** menu (Figure 3-2).

The menu you are presented with contains the most common views in the context of the current perspective. If the view you want to add is not listed in the menu, select the **Other** option and a dialog will be displayed listing all available views to choose from. The views are grouped according to the roles, (perspectives), in which they are usually needed. You might have to expand the groups and scroll the list to find the view you require. Confirm your selection by clicking the **OK** button.

Should the view you have selected already exist in the perspective, it will be brought to the top and made active. If you select a new view it will be added to your perspective and will made active.

To remove a view from a perspective, you simply need to close it using the **Close** icon  in its top right corner.

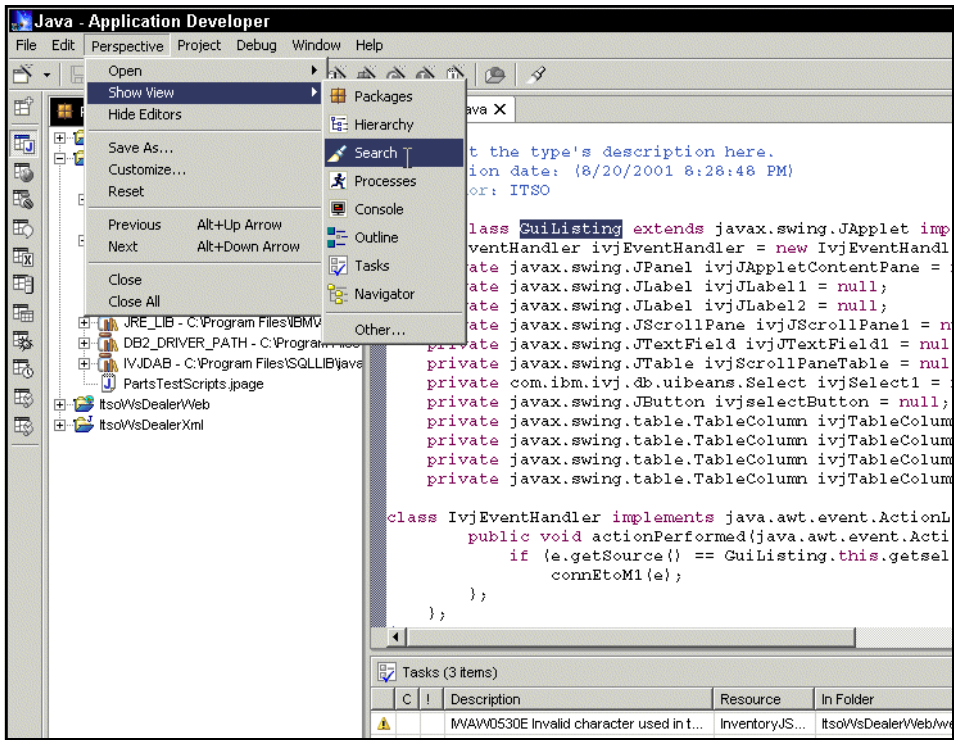


Figure 3-2 Adding a view to a perspective

3.2 Resource perspective

The default Application Developer perspective is the Resource perspective (Figure 3-3).

If a different default was not selected during the installation, the Resource perspective will be displayed the very first time you start Application Developer and any time you restart it after having closed all the other perspectives before exiting. This perspective is always shown at the top of the list when you select **Perspective—>Open** from the menu. Application Developer lets you to change the default perspective. See “Changing the default perspective” on page 39.

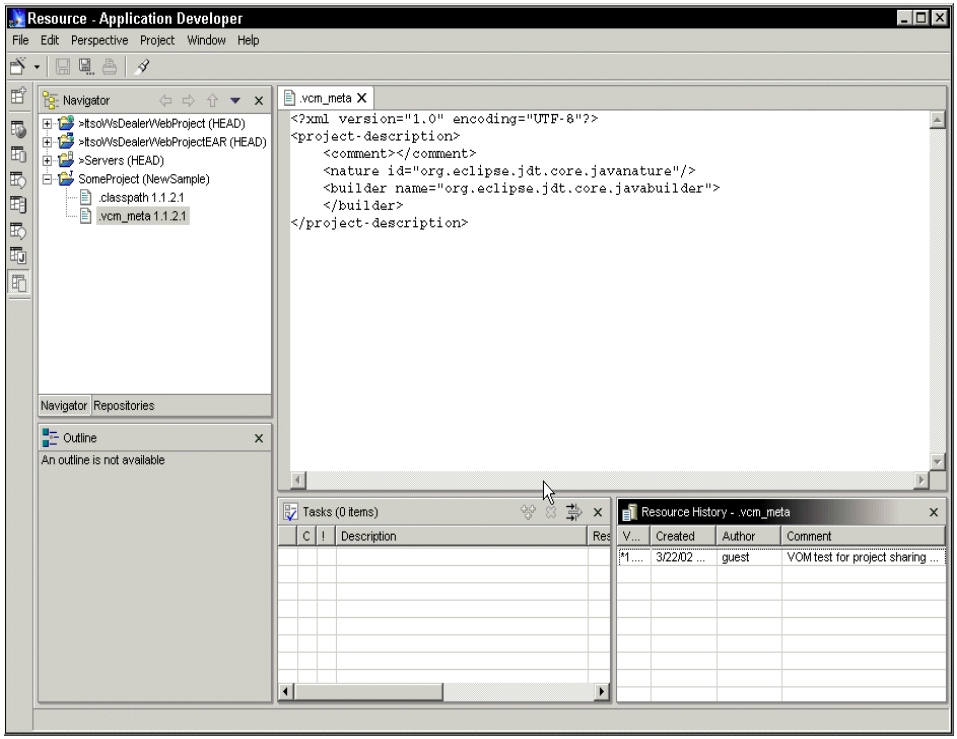


Figure 3-3 Resource perspective

The Resource perspective contains four panes:

- **Top left**: Shows Navigator and Repositories views.
- **Top right**: Reserved for editors of the selected resource.
- **Bottom left**: Shows Outline view of the resource opened in the active editor.
- **Bottom right**: Shows Tasks view and Resource History view if it exists.

3.2.1 Hierarchy levels

Resources are stored and displayed in the workbench in hierarchies. The following terms are used when referring to resources that are stored and displayed in a hierarchical structure:

- **Root**: Refers to the top level of the workbench contents in the file system.
- **Parent resource**: Refers to any resource that contains another resource.
- **Child resource**: Refers to any resource that is contained within another resource. This term is analogous to "child directories".

Note: Only projects and folders can be parent resources. Only files and folders can be child resources.

The Resource History View provides a list of all the versions of a resource in the repository. From this view you can compare two versions, add a version to the workbench, or open an editor on a version.

Workbench resources have properties associated with them, such as the file name, last modified date, file system path, and file size. Properties that are associated with any given resource depend on the resource's file type. These properties are displayed in the Properties view for the currently selected resource.

Resources also have an associated Properties dialog that can be accessed from the context menu. The Properties dialog is used to display information that is more detailed than can be shown in a property view.

To open a file resource, select the file in the Navigator view. From the context menu choose **Open** or **Open With**. Alternatively you can double-click on the resource to open its default editor.

The workbench supports integrated editors and external editors. The integrated editors are those built especially for the workbench and those supported in the WIN OLE document mode. External editors can be any third party editor that is launched in its own separate window and is not integrated into the workbench's editor area.

3.2.2 Changing the default perspective

As was mentioned earlier the Resource perspective is the predefined default perspective. However, you can set another perspective to be the default perspective.

To change the default perspective:

1. Select **Window—>Preferences** from the menu bar.
2. Expand the **Workbench** item on the left and select the **Perspectives** preferences page.
3. From the list of perspectives select the one you want to define as the default.
4. Click **Set Default**. The default indicator moves to the perspective you just selected.
5. Click **OK.**

3.3 Opening another perspective

To open another perspective in the workbench, select the **Open Perspective** icon  in the top left corner or the workbench working area (Figure 3-4).

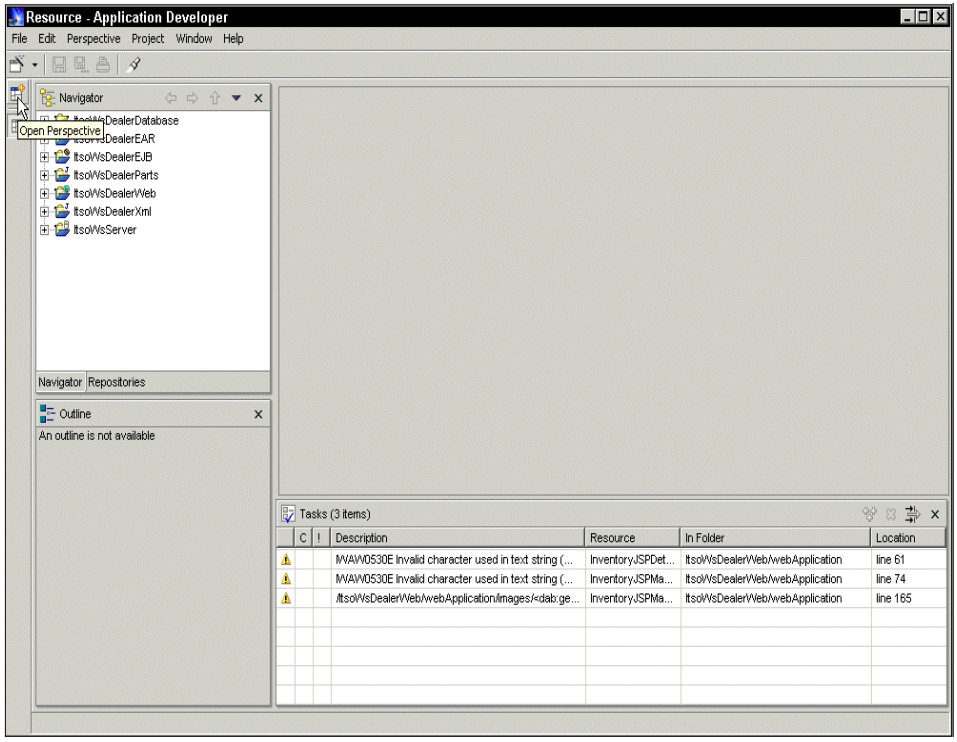


Figure 3-4 Opening a perspective - Step 1

Select the perspective from the menu that is displayed. The most used perspectives are shown as selections on this menu, while the less frequently used ones can be selected from the window that will be displayed when you select the **Other** item from the menu. Since the Java perspective is the perspective we will look at in more detail next, we select **Java** from the menu (Figure 3-5).

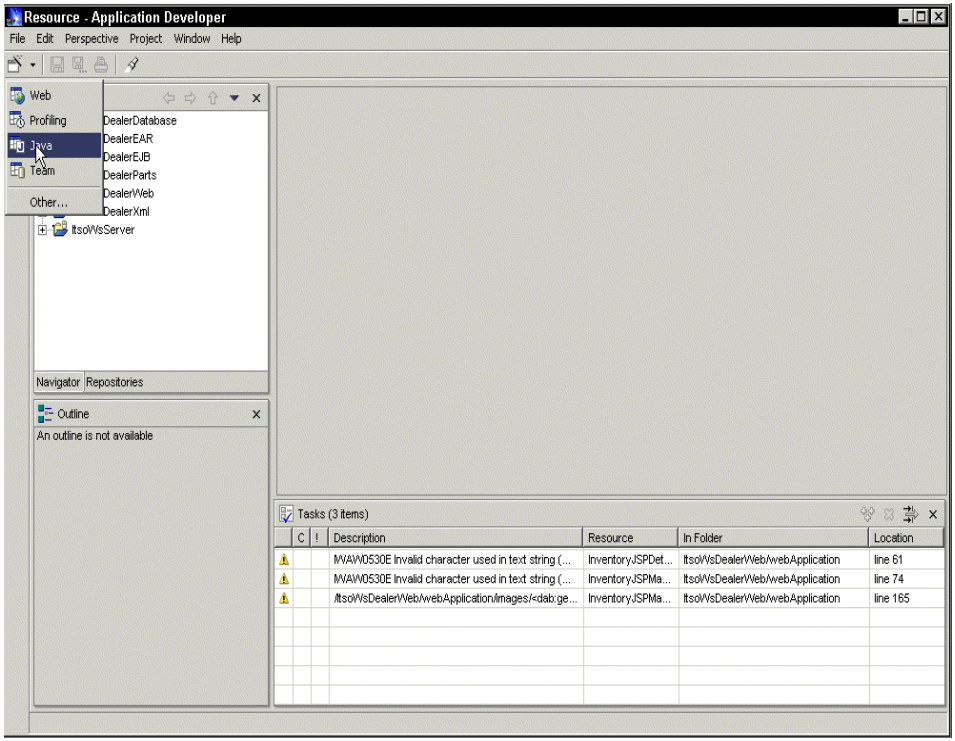


Figure 3-5 Opening a perspective - Step 2

Depending on the workbench preferences, as explained in “Workbench window preferences” on page 31, the new perspective will either be opened in the same window, replacing the current perspective, or in a new window (Figure 3-6).

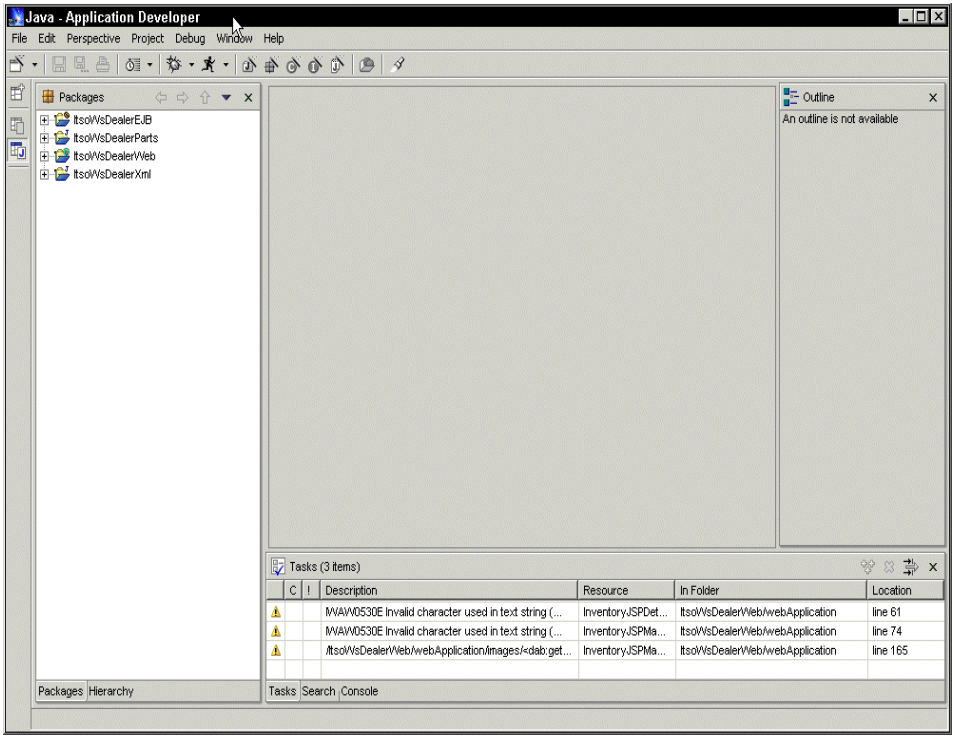

Java - Application Developer
File Edit Perspective Project Debug Window Help
Packages
ItsoWsDealerEJB
ItsoWsDealerParts
ItsoWsDealerWeb
ItsoWsDealerXml
Outline
An outline is not available
Tasks (3 items)
C ! Description Resource In Folder Location
IWAW0530E Invalid character used in text string (... InventoryJSPDet... ItsoWsDealerWeb/webApplication line 61
IWAW0530E Invalid character used in text string (... InventoryJSPMa... ItsoWsDealerWeb/webApplication line 74
/ItsoWsDealerWeb/webApplication/images/<dab:get... InventoryJSPMa... ItsoWsDealerWeb/webApplication line 165
Packages Hierarchy
Tasks Search Console

Figure 3-6 Java perspective opened

Tip: If you want to open a perspective in a new window, you can do this without changing the workbench preferences by simply holding the Shift key down while performing the Open action.

All perspectives that are currently open in a given workbench window are represented by icons in the left most vertical icon bar. You can use these icons to easily switch between perspectives. The currently active perspective is indicated by its icon being shown in the “pushed” state.

You can maximize and minimize the view inside a perspective by double-clicking on its label.

To add more views to a perspective, select **Perspective—>Show View** from the menu bar. You can close, resize, or move any of the views that are currently shown.

3.4 Java perspective

Java developers will use the Java perspective, Figure 3-7, to edit and build their Java code.

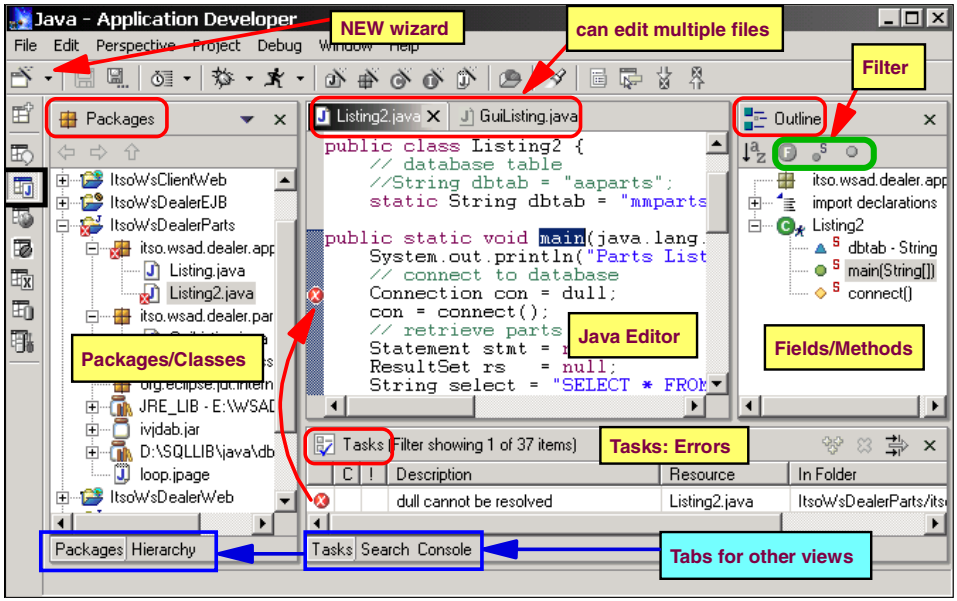


Figure 3-7 Java perspective

The Java perspective contains four panes by default:

- **Left**: Shows Packages and Hierarchy view.
- **Middle**: Reserved for Editors. Multiple files can be edited at one time.
- **Right**: Shows Outline view of the file currently in the active editor.
- **Bottom**: Shows the Task view for error messages and user tasks, the Search view for result of search operations, and the Console view for program output.

The following additional icons are available in the workbench tool bar when the Java perspective is active:

- **New**: Creates new resources or run tools.
- **Debug**: Runs the selected class in debug mode.
- **Run**: Executes the selected class.
- **Create**
  - **Java Project**: Creates a new Java project.

- **Package**: Creates a new package in the selected project.
- **Class**: Creates a new class.
- **Interface**: Creates a new interface.
- **Scrapbook**: Creates a new scrapbook page.

- **Open a Type in the Editor**: Opens any type in the editor.
- **Search**: Opens a search dialog.
- **Edit icons**: Allow you to restrict to displayed source, switch on/off the Hoover Help and provide you the ability to scroll through problems detected in the code. These icons are displayed only when an editor is opened and active.

Tip: If you position the cursor over any icon and leave it there for a short while, a hover help window will open to explain the function represented by the icon.

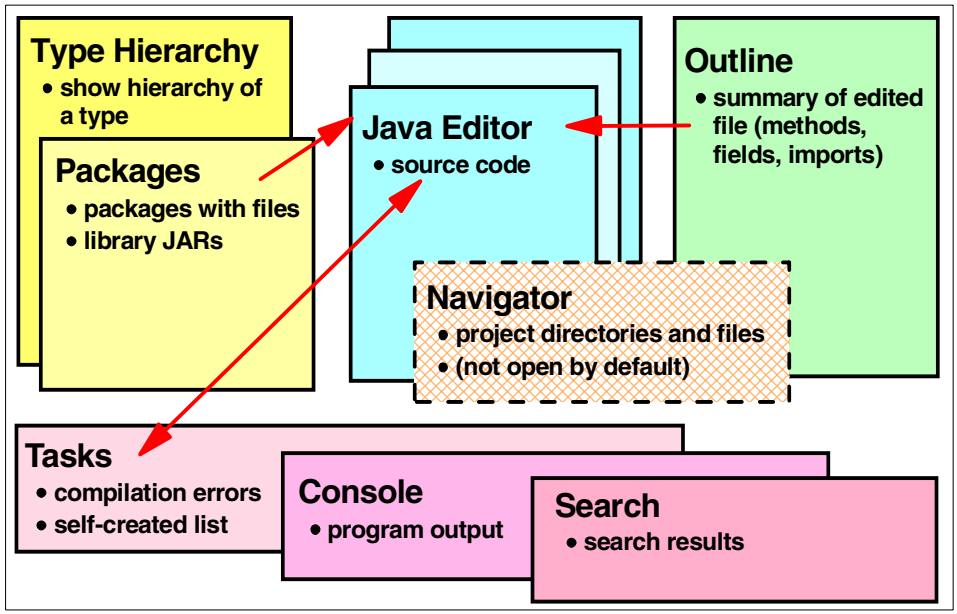


Figure 3-8 Java perspective organization

The following views are supported for the Java development (Figure 3-8):

- **Packages view**: This is displayed by default in the Java perspective and shows the Java element hierarchy of all the Java projects in your workbench. It provides you with a Java-specific view of the resources shown in the Navigator. The element hierarchy is derived from the project's build class

paths. For each project, its source folders and referenced libraries are shown in the tree view. From here you can open and browse the contents of both internal and external JAR files.

- **Navigator view**: Shows the resources organized by folders and sub folders. This view is not displayed by default but can be added.
- **Hierarchy view**: Can be opened for a selected type to show its super classes and subclasses. It offers three different ways to look at a class hierarchy:
  - **Type Hierarchy**: Displays the type hierarchy of the selected type, that is its position in the hierarchy, along with all its Superclass and subclasses).
  - **Supertype Hierarchy**: Displays the supertype hierarchy of the selected type.
  - **Subtype Hierarchy**: Displays the subtype hierarchy of the selected type.
- **Search view**: Shows the results of search operations.
- **Console view**: Shows the output produced when executing programs. Any System.out.print calls will write to this view.
- **Tasks view**: Shows a list of compilation errors or warnings as well as any user-created tasks. Double-clicking on a Java problem opens the editor with the Java source code file and positions the cursor at the line within the source where the problem was detected.
- **Editor windows**: Show Java source code.
- **Outline view**: Shows the elements (imports, class, fields, and methods), that exist in the source file that is currently open in the editor. Clicking on an item in the outline will position you in the editor view at the line where that structure element is defined.

3.4.1 Type Hierarchy view

You can open the Hierarchy view for any type by selecting the menu item **Open Type Hierarchy** from the context menu of a selected type. This can be done both from the Packages and the Outline view (Figure 3-9).

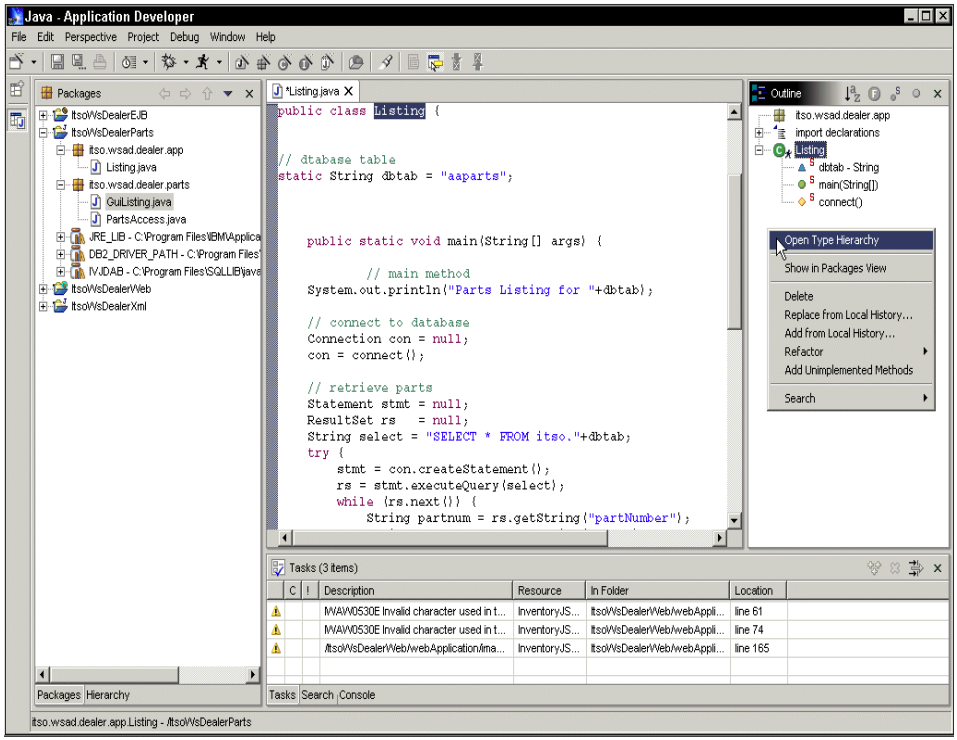


Figure 3-9 Opening hierarchy view from outline view

The content of the top left pane in the Java perspective changes to display the hierarchy of the selected type (Figure 3-10).

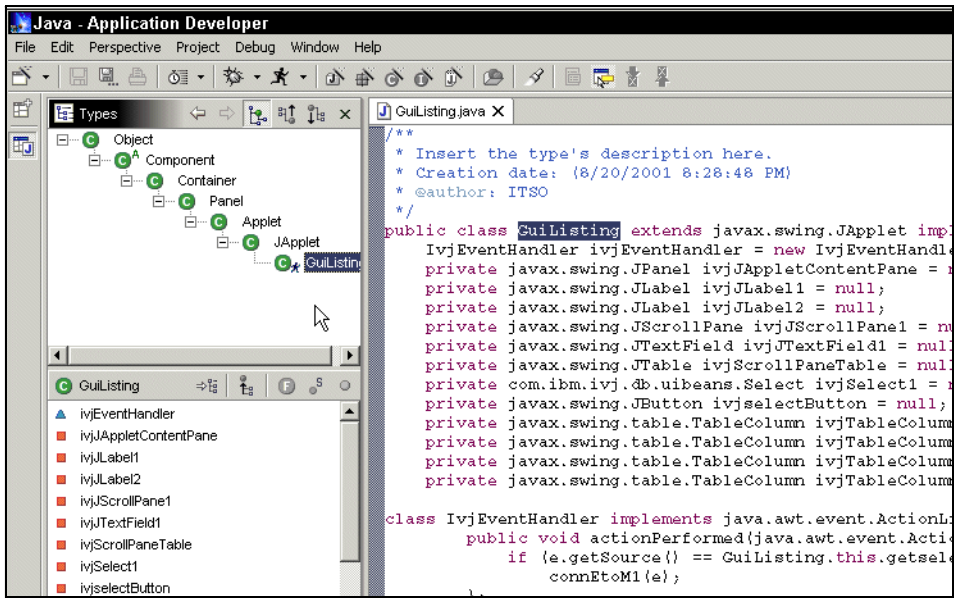


Figure 3-10 Hierarchy view

Icons are provided at the top of the Hierarchy view to display the hierarchy top-down (type hierarchy  and subtype hierarchy ), or bottom-up (super type hierarchy ); seeFigure 3-11.

The supertype hierarchy also shows interfaces that are implemented for each class in the hierarchy.

The Lock View icon  can be used to show from which classes a selected method is inherited. In our example you can see that the `init` method, which is selected in the `GuiListing` class, is inherited from the `Applet` class.

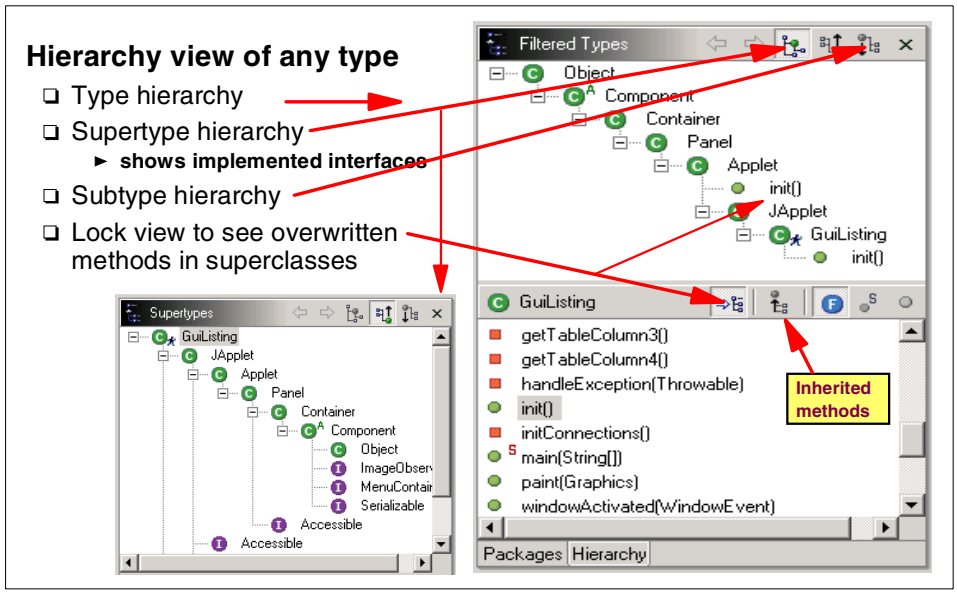


Figure 3-11 Type hierarchy view

Many system wide preferences for Java development are defined in the preferences dialog (**Window—>Preferences**). These include:

- Defining variables to refer to external JAR files instead of specifying physical directory locations. Several variables that point to JAR files in Application Developer itself are predefined.
- Code formatting rules.
- Installed Java runtime libraries. Each project can be associated with a particular runtime library.
- Font used by the Java editor.
- Organization of the import statements that can be generated by the Java editor.
- Refactoring defaults.

More details about these preferences can be found in Chapter 2, “Setting up your workbench” on page 23.

3.4.2 Java code editor

Application Developer provides an editor that has specific support for editing Java code. By default, this editor is activated by either:

- Selecting **Open With—>Java Editor** from the context menu of a selected Java file in the Navigator or Package view.
- Double-clicking a Java file in the Navigator or Package view.
- Single-clicking a currently open Java file in the Navigator or Package view. This method is available only when the **Link Navigator Selection to Active Editor** option is turned on (**Window—>Preferences—>Workbench**)
- Selecting **Open** from the context menu of either a type or a method in the Hierarchy view.

The Java editor works in conjunction with the Outline view. Selecting an item in the Outline view positions the cursor in the editor window to that part of the source code. You can choose to show only the selected item in the editor window rather than the whole source file. There are also icons  that allow you to filter out fields, static, or public members from the Outline view.

The Java editor includes the following features:

- **Syntax highlighting**: Colors are used to highlight keywords, strings, and comments.
- **Content assist/Code assist**: Positioning the cursor over a variable or method displays its definition as a hover help text. Pressing **Ctrl-Space** invokes the code assist feature, which displays possible method calls that can be inserted at the current cursor position Figure 3-12.
- **Code formatting**: Your personal or project code formatting preferences can be set on the Appearance page (**Window—>Preferences** and select **Java** and then **Code Formatter**).
- **Import assistance**: Pressing **Ctrl+Shift+M**, (or selecting a type in source and selecting **Edit—>Add Import** menu item or **Add Import** from the context menu), adds the required import statements for a selected type inside a compilation unit.
- **Integrated debugging features**: Debugging is an integral part of the Java Code editor.
- **Changeable scope**: The Java editor can be configured to show either an entire compilation unit or a just a subset of Java elements.

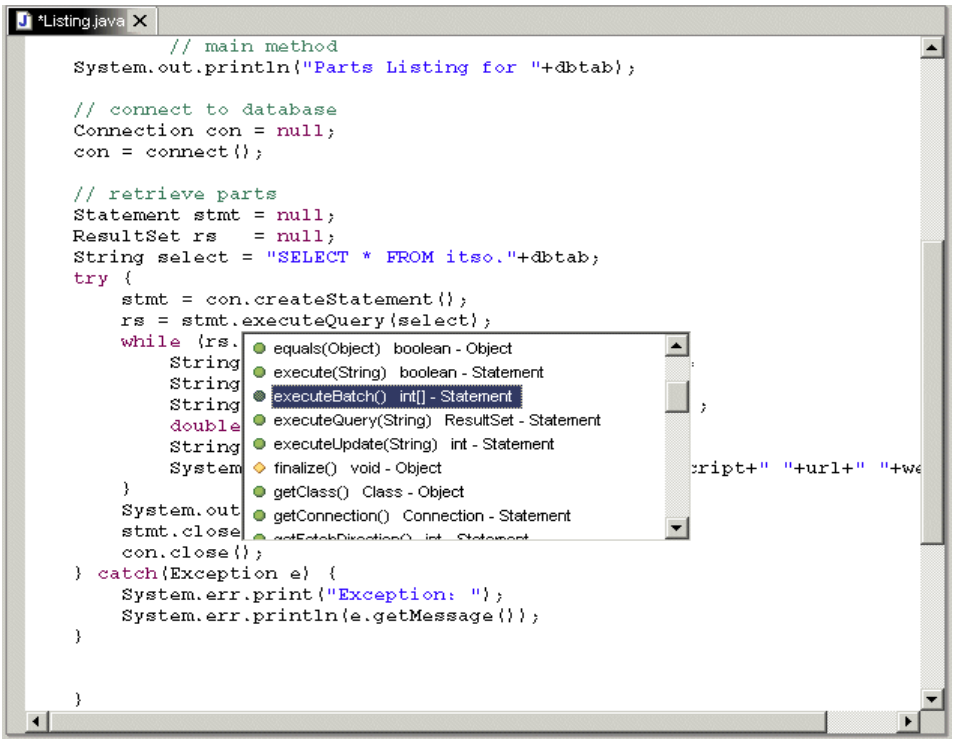


Figure 3-12 Code assist feature

Bookmarks

Bookmarks can be assigned to any line in a file or to a whole file. This is done by selecting the **Add Bookmark** context menu item in the Packages view for a whole file, or by selecting **Add —>Bookmark** from the context menu in the editor for a specific line.

From the Bookmarks view you can then later select the bookmark to open the file and jump to that particular line.

Since the Bookmarks view is not by default included among the views in the Java perspective, you must add it to the list of views.This is done by selecting the Bookmark view from the list of views displayed after selecting **Perspectives—>Show—>Other** (Figure 3-13 and Figure 3-14).

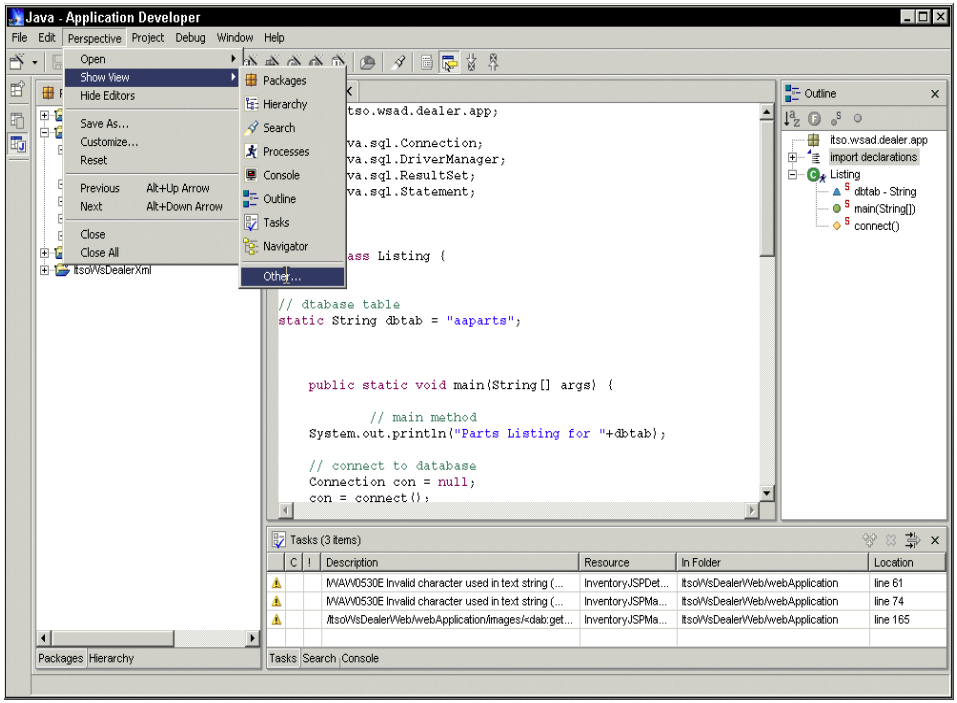


Figure 3-13 Adding bookmark view to Java perspective

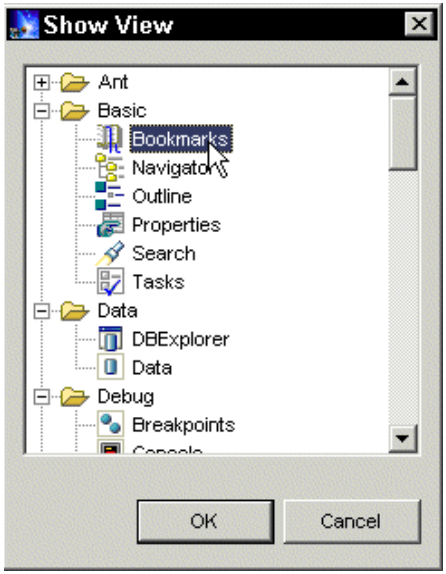


Figure 3-14 Selecting bookmark view

Search

Two types of search are supported (Figure 3-15):

- **Text search**: Scans files for strings matching a search string.
- **Java search**: Searches only for Java constructs (types, constructors, methods, fields). You can limit the search to **Declarations** or **References** or you can search for **Both**.

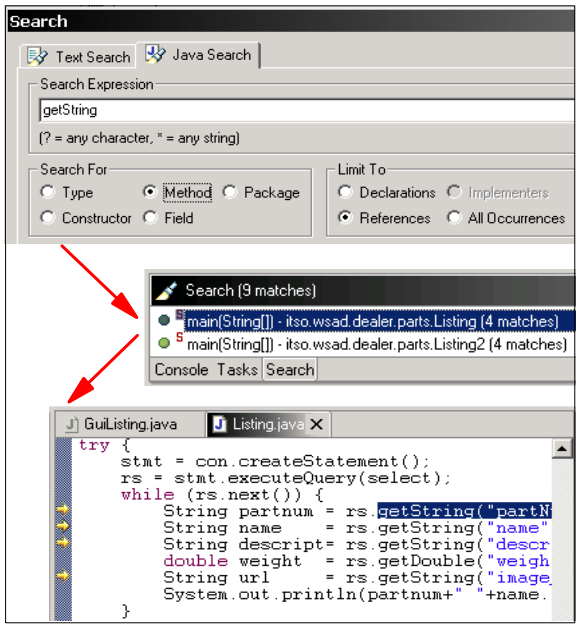


Figure 3-15 Text and Java search are supported

Search results are displayed in the Search view. If you double-click on one of the result files it will open in the editor with yellow arrows showing the lines where a match was found.

3.5 Web perspective

Web developers can use the Web perspective, Figure 3-16,to build and edit Web resources, such as servlets, JSPs, HTML pages and images, as well as the deployment descriptor file, web.xml.

The Web perspective contains four panes:

- **Top left**: Shows the Navigator view, that displays the folders and files of the project.

- **Top right**: Reserved for editors.
- **Bottom left**: Shows the Outline view for the active editor or the Gallery for HTML and JSP files.
- **Bottom right**: Shows Tasks, Properties, Links, Thumbnail, Styles, Color, Palette views.

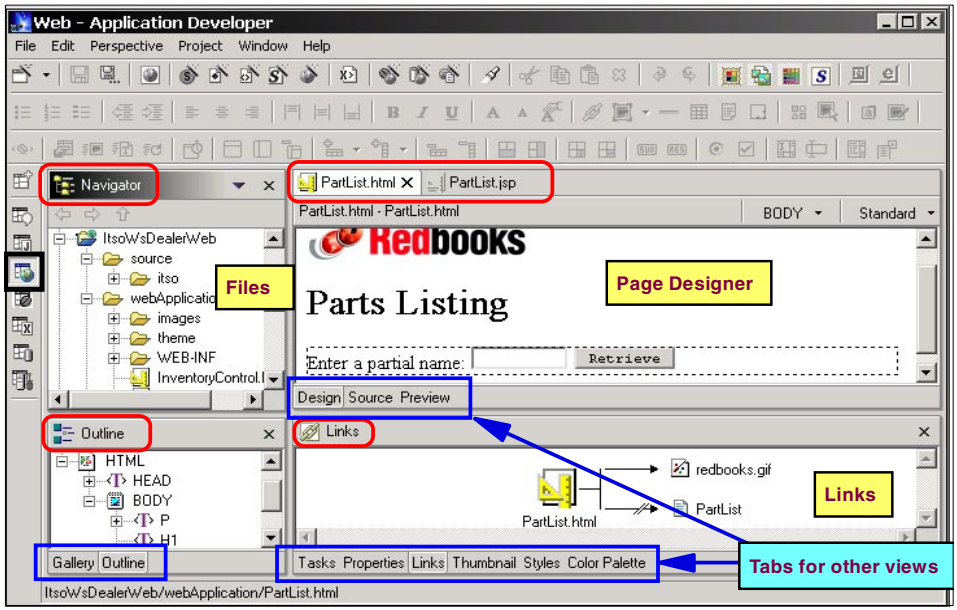


Figure 3-16 Web perspective

The following additional views are supplied to support Web application development:

- **Gallery view**: Contains a number of catalogs with reusable files that can be applied to Web pages. The available file types include images, wallpapers, WebArts, sound files, style sheet files, and JavaScript files.
- **Outline view**: Shows the outline of the file that you are currently editing. For example, for an HTML file, the Outline view shows the tag structure and hierarchy within the HTML file. The context menu for any selected tag enables you to remove the tag, add an attribute to the tag, (if any exist or have not already been specified), add a child tag, and add a sibling tag before or after the tag.
- **Properties view**: Provides options for changing the appearance and state of objects in your file. The properties view is an expandable table that maps properties to their values.

- **Links view**: Shows the resources that the selected file in the Navigator view links to or uses. It also shows the files that link to the file selected in the Navigator view or open in Page Designer.
- **Thumbnail view**: Shows thumbnails of the images in the selected project, folder, or file. This view is especially valuable when used in conjunction with the Gallery view to add images from the artwork libraries supplied by Application Developer to your page designs. You can drag and drop from this view into the Navigator view or the Design view of Page Designer.
- **Styles view**: Provides guided editing for cascading style sheets and individual style definitions for HTML elements.
- **Colors view**: Allows you to apply colors from a palette as well as custom colors to selected objects in the editing area.

3.5.1 Page Designer

The Application Developer Page Designer, which is similar to the one that was available in WebSphere Studio classic, is the default editor for HTML and JSP files in Application Developer.

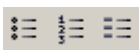
Page Designer gives you with three different views of your HTML page or JSP:

- **Design view**: Supports WYSIWYG construction.
- **Source view**: Shows the HTML source.
- **Preview view**: Shows the page as it will appear in a browser.

Additional tools provided are the Animated GIF Designer, the WebArt Designer, and the Stylesheet editor.

When you are working with Page Designer a number of additional icons are added to the toolbar, providing you with shortcuts to the most common tasks you will perform when developing Web resources:

- **Create**:
  -  Web Project
  -  Java Servlet class
  -  HTML file
  -  JSP file
  -  CSS file
  -  Image file
  -  XML file
-  Create Web Pages from JavaBean

-  Create Web Pages that access and display database fields
-  Edit operations (enabled when you have something selected/done in Source view):
  -  Cut
  -  Copy
  -  Paste
  -  Delete
  -  Undo/Redo
-  Change attributes (enabled with element selected in the Design view)
-  Edit image (enabled with image selected in the Design view)
- Insert HTML elements (enabled on both Design and Source pages):
  -  Lists
  -  Image File
  -  Link
  -  Table
  -  Form
  -  Layout Frame
  -  Bean
  - and so on
- Others:
  -  Indentation
  -  Alignment
  -  Font manipulations

Design

The Design view of Page Designer is the WYSIWYG mode for editing HTML and JSP files. In this view you can build your Web pages without having to deal directly with the complexity of tagging syntax, navigation, and formatting.

Although many tasks can be performed in the same way in the Source view, the Design view gives you full access to the Page Designer menu options, context menu actions, view-specific GUI options, (such as those in the Styles view), and drag and drop behavior.

The Design view also provides support for frames and page layout, which enables you to easily create more complex Web pages. You can immediately see the impact of different design decisions and you can change the composition and attributes of pages, tags, images, effects, and so forth more efficiently and precisely.

Many actions available through the Page Designer menus are also available from design element context menus, which you access by right-clicking on the element.

Source

The Page Designer source editing capability enables you to directly edit an HTML or JSP as you would in a normal text editor. Any changes you make in the Source page are directly reflected in the Design page. If the change involves the addition or removal of a tag, it will also be reflected in the Outline view. If you add or update an attribute value in the Source page and the Properties view is visible, the Properties view will refresh as well.

The Source view has a number of powerful text editing features, such as:

- **Syntax highlighting:** Each tag type is highlighted differently, enabling you to easily find a certain kind of tag for editing. In addition, syntax highlighting is valuable in locating syntax errors.
- **Unlimited undo and redo**: These options allow you to incrementally undo and redo every change you have made to a file during the entire editing session. For text, changes are accumulated one character or set of selected characters at a time.
- **Content assist**: Helps you to finish tags or lines of code and insert macros. Choices available in the content assist list are based on tags defined by the tagging standard specified for the file being edited.
- **User-defined macros**: You can access user-defined macros (using content assist) to help you quickly add regularly-used tagging combinations.
- **Element selection**: Based on the location of your cursor, (or selection in the Outline view) the element selection indicator highlights the line numbers that include an element in the vertical ruler on the left area of the Source page.
- **Context menu options**: From the editor's context menu, you have many of the same editing options that are available in the workbench **Edit** menu.

There are several ways you can enter, insert, or delete tags and text in the Source view:

- Type in the tags directly
- Use content assist to pick from a list of valid tags
- Select menu items

- Select toolbar buttons
- Change tags using the Properties view

There is often more than one way of performing a specific task. For example, if you want to insert a new table on your page you can accomplish this in the following different ways:

- Use content assist and select the table macro (either the default table macro or a custom macro that you have created) from the content assist list.
- Type in the new tags into the file.
- Copy and paste table tags from the current file or another file.
- Combine the preceding steps to create the table you want.

To edit an HTML or JSP file in the Source view, follow these steps:

1. Open the HTML or JSP file that you want to work with in Page Designer. You may need to click the **Source** tab. Normally you will open the file by double-clicking it in the Navigator view.
2. Edit the code as necessary, using any of the available features. As you move the cursor within your file, or select items from the Outline view, the element selection indicator will highlight the range of lines that make up the element. You can use smart double-clicking behavior. If your cursor is positioned over an attribute value, one double-click selects that value, another double-click selects the attribute-value pair, and a third double-click selects the entire tag. This makes it easier to copy and paste commonly used pieces of code.
3. At intervals you may want to format individual elements, or the entire document, to restore element and attribute indentation to see nesting hierarchies more clearly in the file.
4. Save the file.

Note: The HTML Syntax Validator included in Application Developer validates HTML basic syntax and HTML DTD compliance for HTML and JSP files created in Page Designer.

Outline view

The Outline view provides you with a hierarchical view, or roadmap, of a document. This view is especially useful for navigating complex documents, because as you select an element in the Outline view, the corresponding tags in the Page Designer Source and Design pages will be identified by the **active** tagging. In addition, the Outline view is synchronized with the workbench Properties view, so that the attributes and their values for the selected tag are displayed.

When you start typing in the source page, the Outline view selects the tag that you are working on.

If you need to add child or sibling tags for a element, you can select that tag in the Outline view, and then select **Add Child** from the context menu to add a child tag if any exist, or select **Add Before** or **Add After** to add a sibling tag if any apply.

Properties view

The left column of the Properties view, Figure 3-17, contains a list of attributes and the right column contains the editable attribute values. For example, when you select an image tag 'img', the width and height values appear for editing. In addition, any other attributes allowed by the standard specified in the DOCTYPE for the file (such as the HTML 4.0.1 DTD) are listed. When you click on the value column of a property, you can select from among the list of available attribute values.

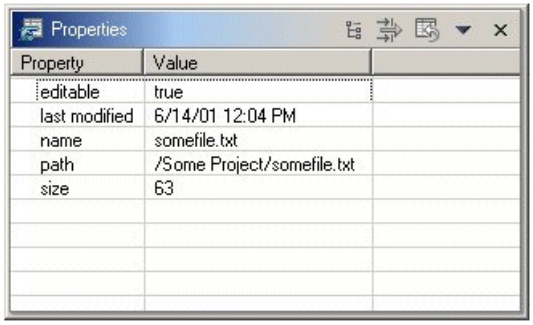


Figure 3-17 Properties view

The context menu in the Properties view enables you to **Undo**, **Cut**, **Copy**, **Paste**, **Delete**, and **Select All**.

Use the **Restore Default Values** button to change any value back to the default setting.

Once you are in the Properties view, you can modify other property values for specific tags. To edit another tag using the Properties view, you can either select the desired tag in the Outline view or just move the cursor within the editing area to the tag you want to edit. The appropriate properties and values are displayed in the Properties view.

Preview

The Preview view shows you how the current page will look when viewed in a Web browser (by default Microsoft Internet Explorer). To preview any dynamic content (such as JSPs), you must use the **Run on Server** option from the page's context menu in the Navigator view.

You can also use the **Tools—>Web Browser** menu option to open the currently selected file in either Microsoft Internet Explorer or a specified version of Netscape Navigator.

JSP Wizards

Two special icons, **Create Web Pages from JavaBeans**  and **Create Web Pages that access and display database fields**  , activate two wizards that are provided in Application Developer to generate skeleton Web applications:

- **Database wizard**: Generates a Web application based on an SQL statement.
- **JavaBeans wizard**: Generates a Web application based on a Java bean.

In the Database wizard, two different models are supported:

- **View bean model**: Follows the MVC pattern and generates a controller servlet, a Java bean for processing, and JSPs for output.
- **JSP taglib model**: Generates a controller servlet and JSPs for processing and output.

The JavaBeans wizard supports only the View bean model.

Each wizard guides you through a series of steps where you specify things like the output folder, whether results should be stored in the session or the request, the model to be used, the SQL statement or Java bean to base the application on, and which output files that should be generated. Once you have completed your selections, Application Developer will generate all the required HTML elements and Java classes for you.

3.5.2 Web.xml editor

The Web application deployment descriptor is the *web.xml* file. This file contains information about the servlets and JSPs in your Web application along with additional deployment information. A special editor is supplied to maintain the web.xml file (Figure 3-18).

The web.xml file is used to build a WAR file from a project and contains the necessary information for deploying a Web application module. Whenever you create a new Web project, Application Developer creates a minimal web.xml file in the WEB-INF folder under the project's webApplication folder.

If you import a WAR file into an existing Web project, you have the option of importing the web.xml file included in the WAR file as the Web project's new deployment descriptor. Any specific deployment information already defined in the file will be used when deploying the updated Web application.

In the course of development, the web.xml file will be updated automatically to reflect changes to your Web project. For instance, when the **New Servlet** wizard is used to create a new servlet in a Web project, it will by default create the appropriate servlet entry in the web.xml file.

Although the editor provides a source view of the XML tagging for the deployment descriptor file, the recommended way of building web.xml files is to use the viewers in the editor. These viewers provide controls to set many Web application parameters related to servlets, paths, variables referenced, security, welcome and error page details, and other general deployment settings. As you specify deployment information, the web.xml editor generates the appropriate tagging into the file.

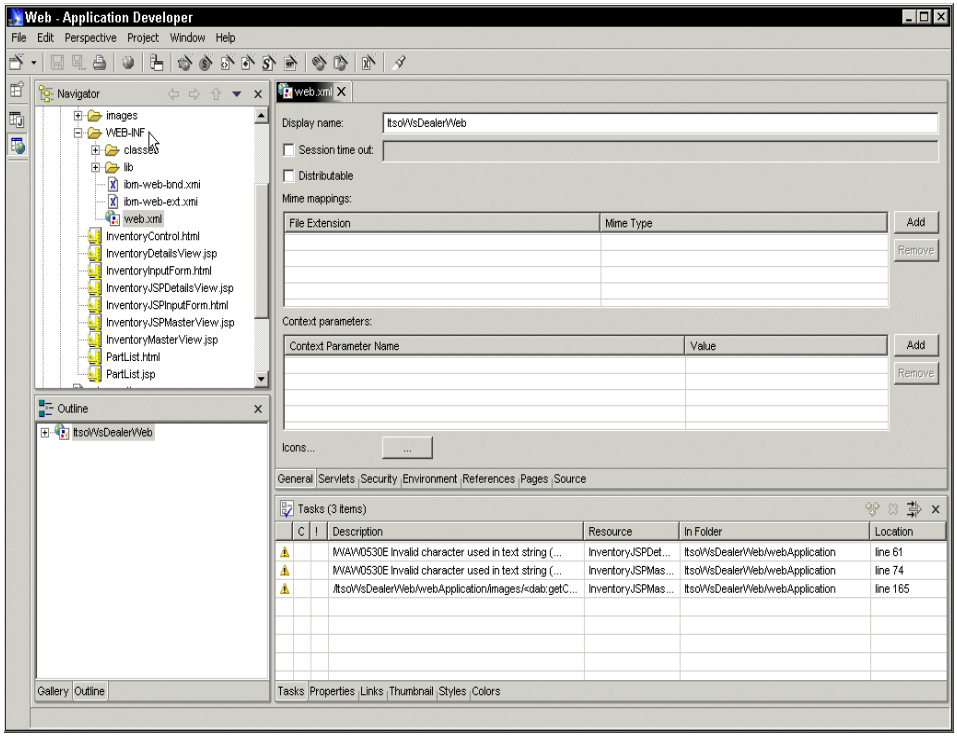


Figure 3-18 Web.xml editor

3.6 J2EE perspective

The J2EE perspective, Figure 3-19, is used for development of EJBs and for the management of J2EE deployment descriptors, (EARs).

Note: EJB development is outside the scope of this book. This chapter introduces the tools available in Application Developer, but they are not discussed further in subsequent chapters.

The J2EE view is the only view in Application Developer where entity and session EJBs can be developed and maintained. This view displays a logical view of the EJBs with their fields, keys, and underlying Java files, (bean class, home and remote interface, key class).

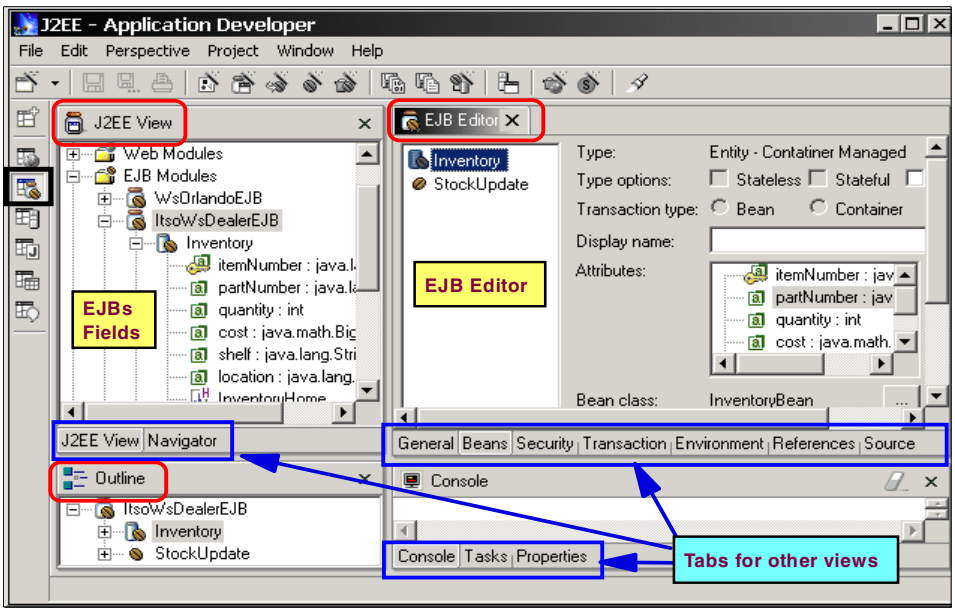


Figure 3-19 J2EE perspective

The J2EE perspective contains four panes:

- **Top left**: Shows the J2EE and Navigator views, that displays the logical structure and resources of the project.
- **Top right**: Reserved for editors.
- **Bottom left**: Shows the Outline view for the active editor.
- **Bottom right**: Shows Tasks view (errors to be fixed) or Properties view of a selected resource.

The J2EE perspective contains the following views that you would typically use when you develop resources for Enterprise Applications, EJBs, Web projects, and Application Client projects:

- **Navigator view**: Provides a hierarchical view of the resources that are in the workbench. These resources reside on your file system, on the workgroup server, or on both. It displays all the project resources (folders and files):
  - The bin folder with the class files
  - The ejbModule folder with the Java source code
  - The META-INF folder with the schema, deployment descriptors, and mapping
- **J2EE view**: Shows the logical structure of the EJB modules and only the main files:
  - The home interface
  - The remote interface
  - The bean itself
  - The key class the holds the key attribute of an entity EJB

  The J2EE view provides you with an hierarchical view of the content modules for resources in the workbench. Use this view to look at the different modules in your projects. By double-clicking on containers in the J2EE view, you can edit your descriptor files.

  There are seven groups in the J2EE view. Each view represents a different part of a enterprise application:
  - **Enterprise Applications**: Shows a hierarchical view of the deployment descriptors for all Enterprise Application projects.
  - **Application Client Modules**: Shows a hierarchical view of the deployment descriptors for all Application Client projects.
  - **Web Modules**: Shows a hierarchical view of the deployment descriptors for all Web projects.
  - **EJB Modules**: Shows a hierarchical view of the deployment descriptors for all EJB projects.
  - **Server Configurations and Server Instances**: Shows a tree view similar to the tree views in the Configuration view of the Server perspective. The same actions are available in both views.
  - **Databases**: Shows database files in any project of the workbench. This view is similar to the Data view used in the Data perspective.

  It is important to note that the J2EE view is an object representation of the projects. Therefore, not all actions are available from the J2EE view, and file-based options, (such as those available from the Navigator view), do not apply (Figure 3-20).

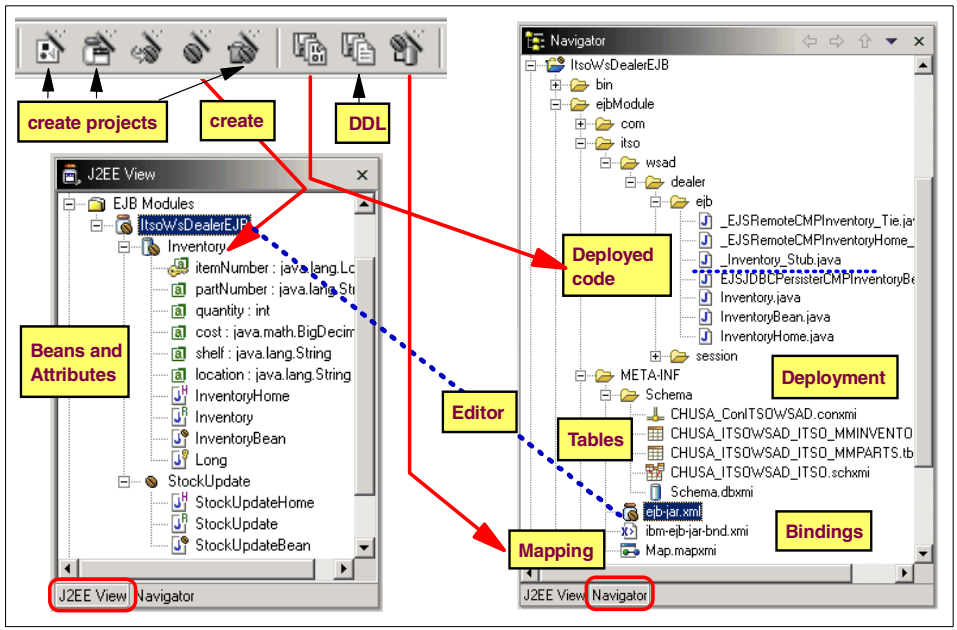


Figure 3-20 J2EE and Navigator views

- **Outline view**: Shows the outline of the file that you are editing. This view changes as you change the open files in different editors or select different editor tabs.
- **Tasks view**: Lists the to-do items that you have entered, plus any automatically logged problems, warnings, or other information associated with the selected project.

  You can double-click on an item to address the specific problem in the appropriate resource.
- **Properties view**: Provides a tabular view of the properties and associated values of objects in files you have open in an editor.

3.6.1 Application editor

You can open an Application editor, Figure 3-21, by double-clicking on your application in the J2EE View or by selecting the **Open With—>Application Editor** menu item from the context menu of the selected application.

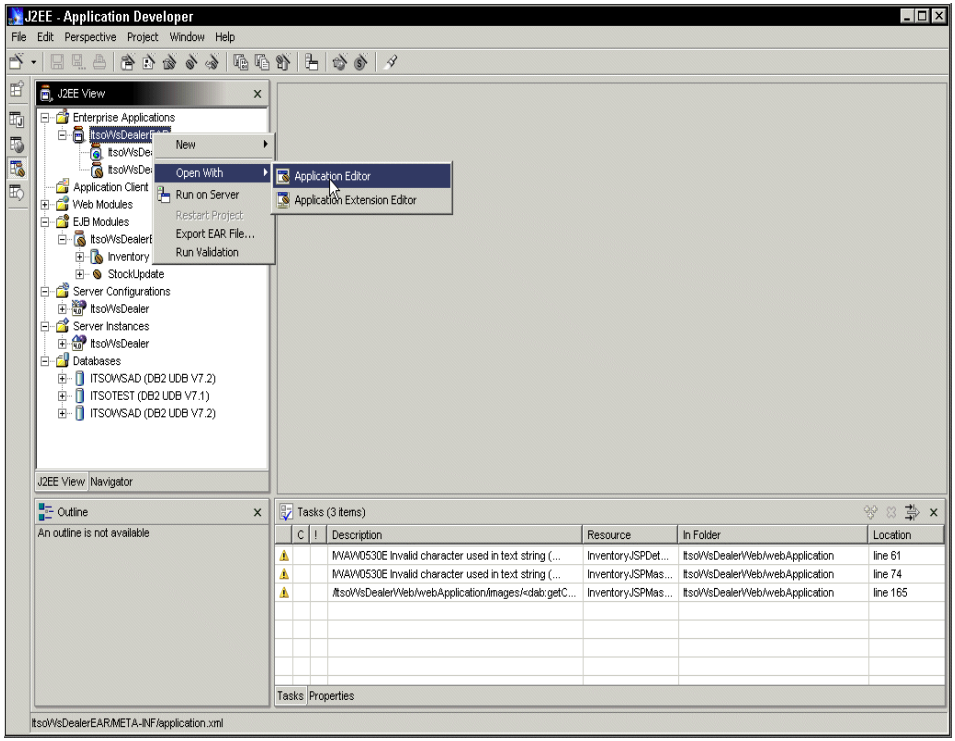


Figure 3-21 Opening the application editor

The Application editor, Figure 3-22, consists of the following pages and views:

- **General page**: You are able to choose the icons that will be used to represent your enterprise application. These icons are mainly used for identification on the server.

  In order to use an icon, you must first import the graphic file into the Enterprise Application project. It must be contained inside the EAR file in order for it to be found at deployment time. Once the file has been imported into the project, you will be able to select it within the icon dialog on the Application editor.

  If you do not import the file into your project, you will not see any icons within the dialogs.

- **Modules page**: Used to add, modify, and delete EJB, Web, and Client modules. When you select a module in the Modules pane, its attributes are displayed in the fields on the right side of the pane.

These fields include:

- **URI**: Stands for Uniform Resource Identifier and is an editable text field. A URI is an encoded address that represent any resource, such as an HTML document, on the Web. As opposed to a URL or URN, which are concrete entities, a URI is an abstract supertype.
- **Context root**: Only applies to Web modules. The context root is the Web application root, that is the top-level directory of the application when it is deployed to a Web server.
- **Alternate descriptor**: Not supported in this version.

- **Project page**: Identifies the project to which the module is associated. Select the Browse button to search for and select a different project.
- **Security page**: Used to view, add, remove, gather, and consolidate security roles.

  Use the **Gather Roles from Modules** button to roll up all security roles defined in modules that are included in the application.

  Use the **Consolidate Roles** button to replace an original role by another, existing role. After consolidation, the original role is removed from the application, and if it is gathered from a module, it will be removed from the module, too.
- **Source page**: Used to view and modify the XML source code. The Source page is an XML editor in itself.

  The XML source changes dynamically when you make changes in the Application editor. Also, when the changes are made in the XML source, the Application editor reflects these changes.

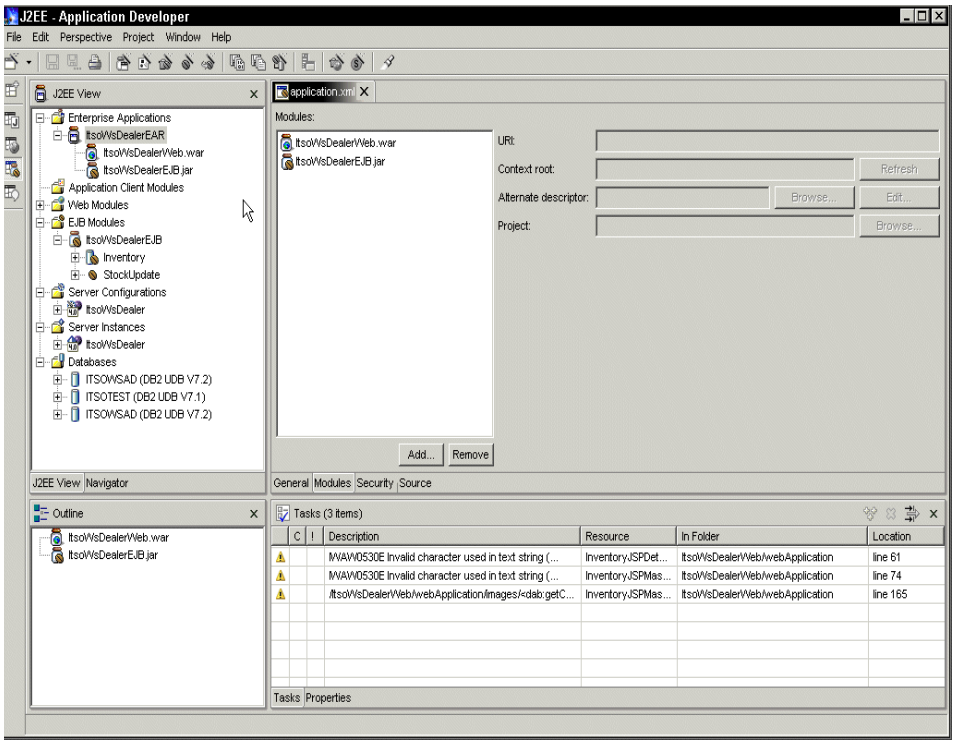


Figure 3-22 Application editor

3.6.2 Application Extension editor

Extensions are additions to the standard J2EE deployment descriptors. They enable the use of enterprise extensions, older systems, or behavior not defined by the J2EE specification.

You can open an Application Extension editor by selecting the **Open With—>Application Extension Editor** menu item from the context menu upon your selected application.

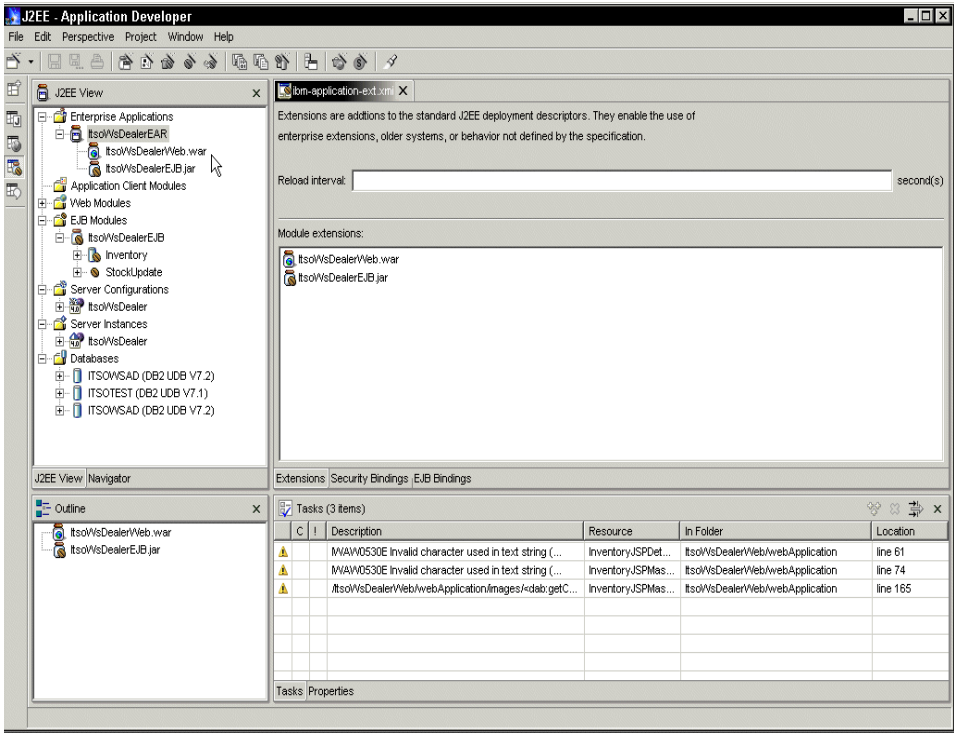


Figure 3-23 Application extension editor

The Application Extension editor, Figure 3-23, consists of the following pages and views:

- **Extensions page**: Used to view module extensions to the deployment descriptors and set the desired reload interval.
- **Security Bindings page**: Used to view, add, remove, and bind objects to an existing security role in the application. When you select a role in the list, you can select from the following options to bind the role:
  - Everyone
  - All authenticated users
  - Users/Groups

  Selecting the Users/Groups checkbox enables adding, removing, and renaming a user or a group security role binding.
- **EJB Bindings page**: Allows you to specify on behalf of which user an enterprise bean will execute at runtime. This only applies if the enterprise bean used the IBM "Run As Role" extension in its deployment descriptor.

The third column of the table contains a drop-down list of role names. This list is only a subset of the roles in the application, and are defined in the EJB Extension editor's Method page as "Run as" mode. The Add button will be enabled only if the drop-down list is not empty. Then you can specify a user ID and password for a role from the list.

Both the Application editor and the Application Extension editor modify a set of files that include:

- application.xml
- ibm-application-ext.xml
- ibm-application-bnd.xml
- module maps

Attention: Changes in the Application/Application Extension editor are not committed until the editor is saved.

If the editor is closed without saving, then the changes are not committed. You will only get prompted to save the Application/Application Extension editor if there are no other references to it or its resources.

If there are two Application/Application Extension editors opened in two different perspectives, or an Application editor and an Application Extension editor are opened, the changes will appear in both and you will only be prompted to save when the last is closed.

The Outline view can be used as a navigational tool, once an object is selected in the Outline view, the Application/Application Extension editor will switch to the corresponding page and highlight the selected object.

Another purpose of the Outline view is to allow the user to quickly add or remove objects. Each object in the Outline view has a pop-up menu that allows you to create or remove new objects. The object can also open on to the Source page where the representing XML tag within the deployment descriptor will be highlighted.

3.6.3 EJB editor

The EJB editor is used to modify the EJB JAR file and the associated Java files.

Attention: The EJB editor and the EJB Extension editor cannot be open on the same resource in the workbench. However, using **Open With...** from the J2EE view allows you to open both editors at the same time. This is also true for the Application editor and the Application Extension editor.

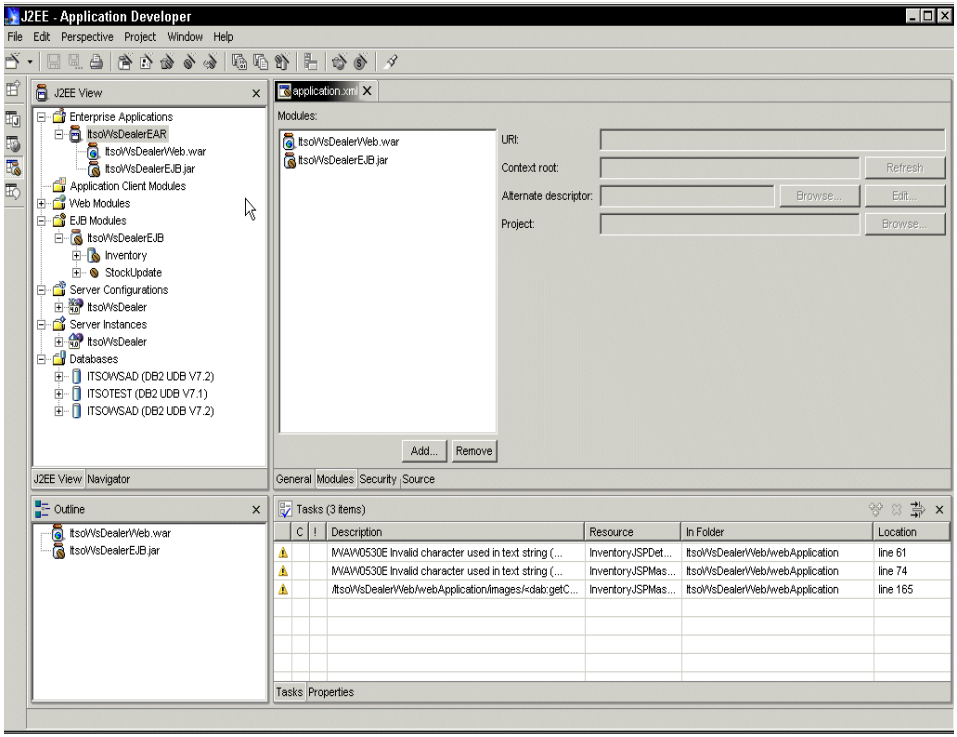


Figure 3-24 EJB editor

The EJB editor, Figure 3-24, consists of the following pages and views:

- **General page**: On this page and on the Beans page you will be able to choose icons that represent your beans. These icons are mainly used for identification on the server.

  In order to use an icon, you must first import the graphic file into the EJB module. It must be contained inside the JAR file in order for it to be found at deployment time. Once the file has been imported into the module, you will be able to select it within the icon dialog of the EJB editor.

  If you do not import the file into the EJB module, you will not see any icons within the dialogs.

- **Bean page**: Used to add, modify, and delete enterprise beans and their attributes. The bean page allows you to access the specialized Java editors. The specialized EJB Java editors can also be opened from the Navigator view by selecting **Open With—>Enterprise Bean Java Editor** from the context menu.

  When opened on the bean class, you can promote methods to the home or remote interface.

The beans page also allows you to change the key class, the remote interface, the home interface, or the bean class for that bean.

Note: The hover help for this Java editor will indicate to which enterprise bean this Java file belongs.

- **Security page**: Used to view and define security roles and method permissions.

  Most of the views in the EJB editor are tables or table trees. In the tables, objects can be edited by clicking on them. In the table trees, the items are grouped by bean name, followed by the editable object.

  On the security page, there is a table and a table tree. These two views work together.

  In order to create a method permission you must first create a security role. When you add a security role you will see a column appear in the method permission table.

  You can now add a method permission by clicking the **Add** button. In the dialog, there will be a list of methods from the home and remote interfaces, plus the common methods between the two. There will also be a list of security roles that you just created.

  Once your method permission is created, there will be a new row in the method permission table. The new row contains the method name, (which cannot be edited), and check boxes. The check boxes will appear in the security roles columns.

  You can edit your permissions by selecting or deselecting the checkboxes.

- **Transaction page**: Used to view and define transaction methods for enterprise beans.

  The transaction page is a table tree. The table tree contains editable cells. These cells are either text cell editors or combination box cell editors. You will not be able to add new values in the combo box since these lists contain the only valid values. For instance, valid transaction types for a bean are:

  - Not supported
  - Supports
  - Required
  - Requires new
  - Mandatory
  - Never

- **Environment page**: Used to add, modify, and delete environment variables.

- **References page**: Used to add, modify, and delete EJB resources. You can switch between editing Resource References and EJB References.
- **Source page**: Used to view and modify the XML source code associated with deployment descriptor for the EJB JAR.

The Source page is a XML editor in itself. The XML changes dynamically when the EJB editor is edited. Also, when the XML changes the EJB editor reflects these changes.

Attention: Changes in the EJB editor are not committed until the editor is saved. If the editor is closed without saving, then the changes are not committed. This also includes the changes that are made to the Java files, and/or the bindings and extension files, which can be edited by the EJB Extension editor.

If an existing Java editor is currently opened and a change is made in the EJB editor that causes code to be generated into the Java file that is currently opened, the changes will be generated into the open editor.

If the EJB editor is saved, the Java changes will also be saved.

If the EJB editor is closed without saving, all changes will be thrown away except the change that is currently in the open Java editor. Those changes will remain because there was another reference to the file that was currently opened. You will only get prompted to save the EJB Editor if there are no other references to it or its resources.

If there are two EJB editors opened in two different perspectives, or you have an EJB editor and EJB Extension editor open in one or more perspectives, the changes will appear in both and you will only be prompted to save when the last is closed.

The Outline view has multiple purposes. It can be used as a navigational tool; once an EJB object is selected in the Outline view, the EJB editor will switch to the corresponding page and highlight the selected object. Another purpose of the Outline view is to allow the user to quickly add or remove objects.

Each object in the Outline view has a context menu that allows you to create new or remove objects. For instance, a CMP entity bean will allow you to add or remove attributes, environment variables, and references.

The pop-up menu also includes the ability to open the specialized Java editors, (if the selection is a bean object). The object can be opened on the Source page where the representing XML tag within the deployment descriptor will be highlighted. There is a pop-up menu called **Quick links** that allows users to bind their bean objects to a JNDI name.

Important: The Outline view on the EJB editor is tailored for advanced users.

3.6.4 EJB Extension editor

Note: The EJB Extension editor is used to modify IBM specific extensions to the EJB 1.1 specification. These artifacts are only used when deployed to **WebSphere Application Server**.

The workflow in the extension editor relies on tree views. Each page has a tree and the selection in the tree enables the editable widgets. The tree of method elements shows all available and existing method elements defined for an enterprise bean. If the icon appears as a yellow warning symbol, it indicates that the defined method element does not resolve to any actual methods.

The EJB Extension editor consists of the following pages and views:

- **Methods page**: Used to define access intent, isolation level, and security identity settings for methods. Security identity settings include the following:
  - Description
  - Run as mode
    - Use identity of caller
    - Use identity of EJB server
    - Use identity of specific role (Role name or Description)
- **Inheritance page**: Used to change the enterprise bean's inheritance structure. The left side of the Inheritance page provides a hierarchical tree view of the EJB inheritance structure. On the right, you define the inheritance characteristics, or set the EJB as a root.

  The bean key class group will only be enabled when changing an entity bean from being an inherited subtype to a root. When you move an EJB from being a subtype to being a root bean, you must specify the key class for it. This is because an inherited entity bean must always use the same key class as its super type root.

  When creating a new key class for CMP EJBs, the fields from the super type will automatically be added.

  When changing the inheritance of a bean, you must select to register changes. The changes will be committed when the editor is saved.

- **Relationships page**: Use this page to add, modify, and remove relationships.
- **Finders page**: Use this page to define Home methods. The Finders page is used to define the metadata for container-managed entity finders for find methods that are already defined on the Home interface.

  The metadata for the ejbql will be automatically generated for you. If you select a finder in the tree view and select the ejbql radio button you will see the a generated ejbql statement. The statement is editable.
- **Container page**: Use this page to define bean cache, locale location, and local transaction settings.

  Bean cache settings include:

  – Activate at
  – Load at
  – Pinned for

  Local transaction settings include:

  – Boundary
  – Resolver
  – Unresolver action

  For further details about the specific Container page settings, please refer to the *WebSphere Application Server documentation*.
- **Bindings page**: Use this page to set JNDI name and data source access permissions. An enterprise bean requires a JNDI name and data source to run on a WebSphere Application Server.

  If the data source is set on the JAR it is not necessary to set the data source for the enterprise beans. The default for the enterprise bean is to use the JAR data source.

  If no JNDI name is specified for the enterprise bean data source, it will use the bean name.

The Outline view is primarily used for navigation. You can use the Outline view to add a relationship role to the primary key of an enterprise bean. Additionally, you can delete relationships in the Outline view.

From the J2EE view, both the EJB editor and EJB Extension editor can be opened using the **Open With** pop-up menus. In the Resource Navigator, you can open the EJB editor on the XML file and the EJB Extensions editor on the ibm-ejb-jar-ext.xmi file.

Note: Changes to the two types of editors, EJB editor and EJB Extension editor, are synchronized and may affect the same set of files. You will only be prompted to save when the last one is closed.

3.7 Server perspective

Application Developer provides support for local and remote test environments for testing of Web applications.

- WebSphere Application Server AEs Test Environment (**A**dvanced **E**dition **s**ingle Server Test Environment) is built into the Application Developer.
- WebSphere Application Server AEs (**A**dvances **E**dition **s**ingle Server) can be installed on the local or a remote machine.
- Apache Tomcat is built into the Application Developer and runs on the local machine. Deployment to remote instance is a manual process.

In order to test a Web application, it has to be published to the selected server by installing the owning EAR project file into the application server. Then the server is started and the Web application can be tested in a Web browser.

If the server runs inside Application Developer, we call it a *local server*. If the server runs outside, on the same or another machine, we call it a *remote server*. A remote server is started through the IBM Agent Controller.

Servers can be defined in Application Developer. To run a server you must have a *server instance* and a *server configuration*. Server instances and configurations are defined in Server projects, which can be shared between developers.

EAR projects, (which contain EJB and Web projects) are associated with server configurations. A configuration can be associated with multiple projects, and a project can be associated with multiple configurations, one of which will be the preferred configuration.

Projects are associated with servers. When a server is started the associated projects are loaded and their code can be executed.

Servers can be started in normal or debug mode. In debug mode, breakpoints can be placed into servlets and JSPs to assist in finding problems.

In the Server perspective you maintain definitions of application servers for testing of Web applications, EJBs, and Web Services. Server configurations define the type of server (WebSphere or Tomcat) and are configured with JDBC drivers and data sources.

Icons are provided to create a server projects  , server instances  and configurations  or both at once  .

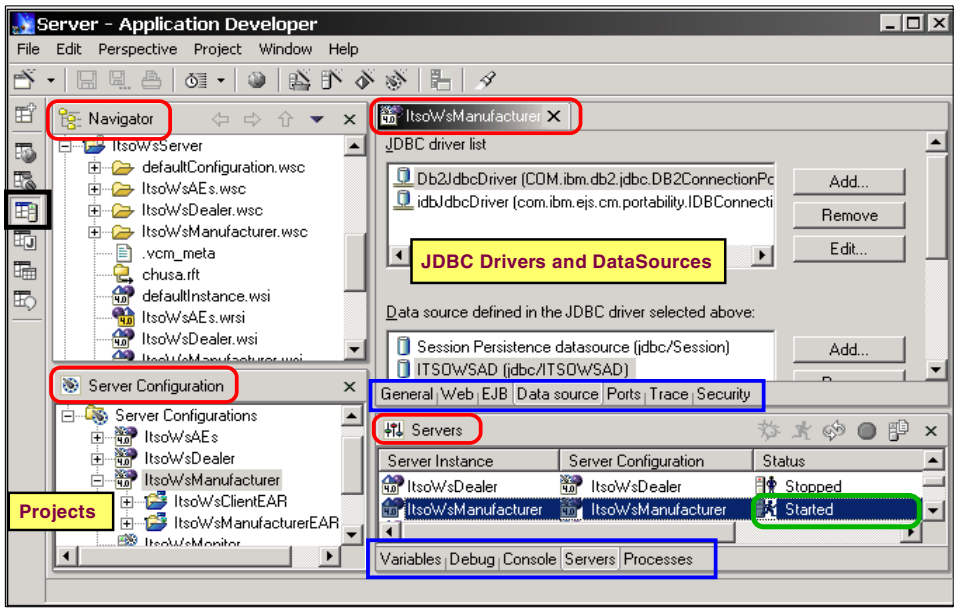


Figure 3-25 Server perspective

The Server perspective, Figure 3-25, contains four panes:

- **Top left**: Shows the Navigator, which displays the logical structure and resources of the project.
- **Top right**: Reserved for editors and browsers.
- **Bottom left**: Shows the Server Configuration view with all defined server instances and their corresponding configurations.
- **Bottom right**: Shows Server Control Panel - the controlling center for server starting, stopping, debugging and tracing.

A server configuration specifies:

Information about the server facilities, such as ports that will be used, JDBC drivers to be loaded, data sources to be defined, mime types, if sessions or cookies are used, and if the universal test client should be loaded.

A server instance specifies:

The runtime environment, PATH and CLASSPATH information, system properties such as the JIT compiler, and how projects will be deployed to a remote server.

A *remote server* requires information about the host address, the installation and deployment directories, and how files should be transferred, (either through LAN copy or through FTP).

A Server project is used to store server definitions. Such a project can be shared and versioned in a team environment. Servers can be defined automatically for simple projects, but in most cases tailored servers will be defined for a set of projects that run on the same server.

A server definition can be stored as a *template* for easy definition of additional servers with same or similar characteristics.

Testing of a Web application involves these steps:

- Define a server and associate the EAR project with the server.
- Start the server in normal or debug mode. You can simply select the project and **Run on Server** to start the preferred server in debug mode, or you can start the server manually.
- Start a Web browser by selecting an HTML file and selecting **Run on Server** from the context menu, or start a Web browser manually, inside the Application Server or outside, and enter the corresponding URL.

To debug a Web application you start the server in debug mode. In this mode you can set breakpoints in servlets and JSPs, step through the code using the standard debug icons, (step into  , step over  , step return ), and monitor the variables.

Debugging Java code is the same as for a Java project. Debugging a JSP is performed at the source code level. You can set breakpoints only at JSP tag lines, not in HTML code. The variables of the JSP servlet can be viewed in the Variables view.

3.8 XML perspective

Application Developer provides a comprehensive visual XML development environment. The tool set includes components for building DTDs, XML schemas, XML, XSLT, and mappings between XML and different backend data stores.

Note: XML development is outside the scope of this book. This chapter introduces the tools available in Application Developer, but they are not discussed further in subsequent chapters.

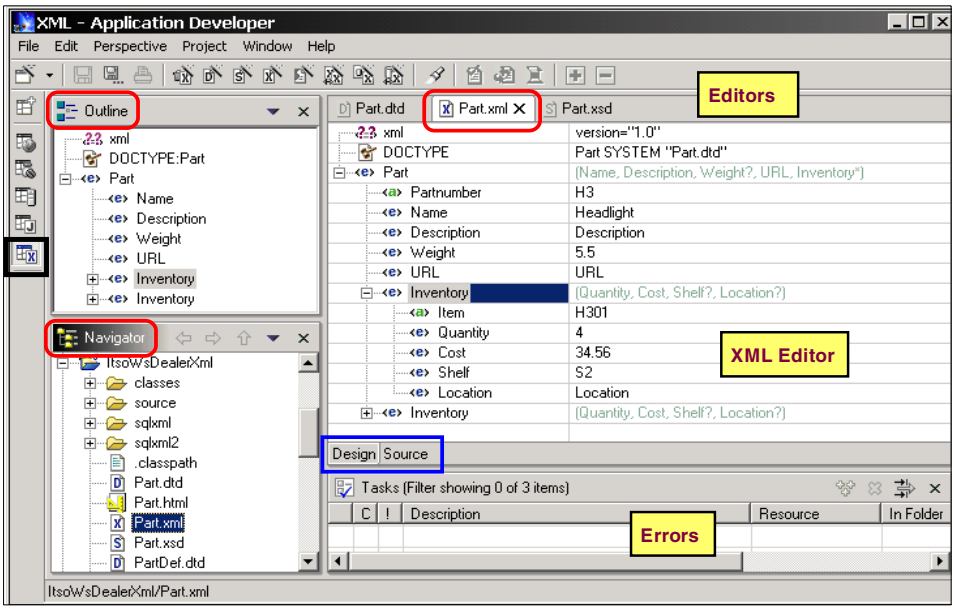


Figure 3-26 XML perspective

The XML perspective, Figure 3-26, contains four panes:

- **Top left**: Shows the Outline view for the active editor.
- **Top right**: Reserved for editors.
- **Bottom left**: Shows the Navigator view, which displays the folders and files of the project.
- **Bottom right**: Shows Tasks view. This view shows problems to be fixed and user defined tasks.

The following XML tools are available:

- **XML editor** is a tool for creating and viewing XML files. You can use it to create new XML files, either from scratch or from existing DTDs or XML schemas. You can also use it to edit XML files, associate them with DTDs or schemas, and validate them.
- **DTD editor** is a tool for creating and viewing DTDs. Using the DTD editor, you can create DTDs, generate XML schema files, and generate JavaBeans for creating XML instances of an XML schema. You can also use the DTD editor to generate default HTML forms based on the DTDs you create.
- **XML schema editor** is a tool for creating, viewing, and validating XML schemas. You can use the XML schema editor to perform tasks such as creating XML schema components, importing and viewing XML schemas,

generating DTDs and relational table definitions from XML schemas, and generating JavaBeans for creating XML instances from an XML schema.

- **XSL trace editor** enables you to visually step through an XSL transformation script, highlighting the transformation rules as they are fired. Additionally, an XSL Trace user can view the XML source and the corresponding XML or HTML result in "real time".
- **XML to XML mapping editor** is a tool used to map one or more source XML files to a single target XML file. You can add XPath expressions, groupings, Java methods or conversion functions to your mapping. Mappings can also be edited, deleted, or persisted for later use. After defining the mappings, you can generate an XSLT script. The generated script can then be used to combine and transform any XML files that conform to the source DTDs.
- **XML and SQL query wizard** can be used to create an XML file from the results of an SQL query. You can optionally choose to create an XML schema or DTD file that describes the structure that the XML file has, for use in other applications. You can also use the XML and SQL Query Wizard to create a DADX file that can be used with the Web services tool. The generated DADX file will contain your SQL query.

  You can use either the SQL wizard or SQL query builder to create the SQL queries your XML or DADX files are generated from.
- **RDB to XML mapping editor** is a tool for defining the mapping between one or more relational tables and an XML file. After you have created the mapping, you can generate a document access definition, (DAD), script which can be run by the DB2 XML Extender to either compose XML files from existing DB2 data, or decompose XML files into DB2 data.

3.8.1 XML editor

The XML editor, Figure 3-27, is a tool for creating and viewing XML files. You can use it to perform a variety of tasks such as:

- Create new XML files from scratch or from existing DTDs or XML schemas.
- Edit and validate XML files.
- Import existing XML files for structured viewing.
- Associate XML files with DTDs or XML schemas.

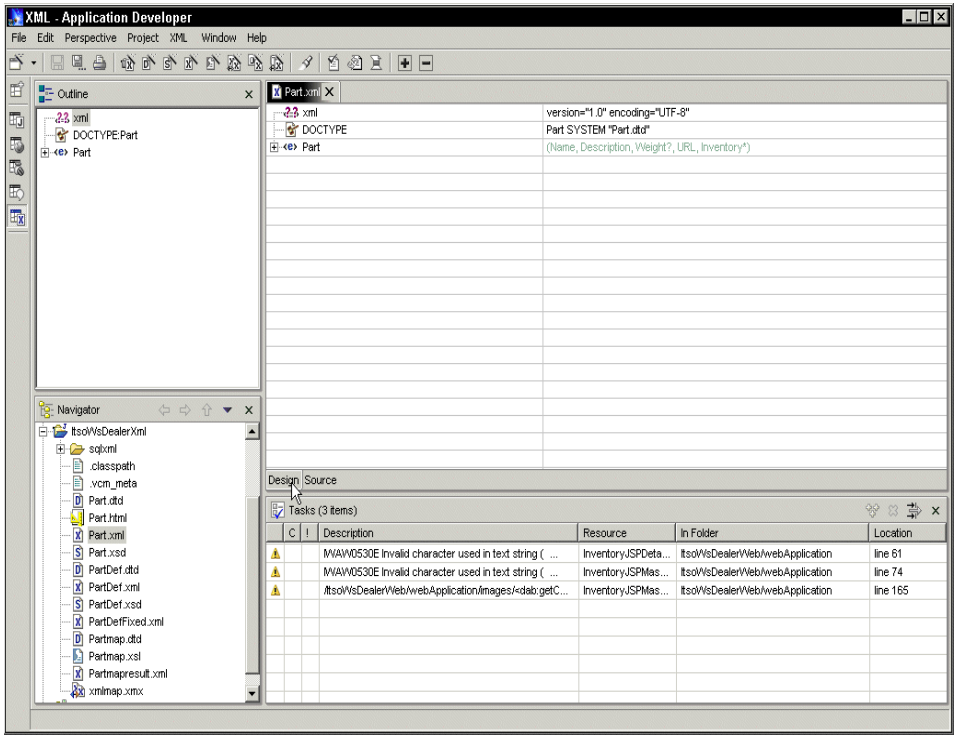


Figure 3-27 XML perspective with XML editor opened

The XML editor provides two views for editing:

- **Source view**: Enables you to view and work directly with the source code file. Many of the XML editing features in the XML editor Source view are very similar to those available in the Source view of Page Designer.

  The Source view has several text editing features, such as syntax highlighting, unlimited undo/redo, and user-defined macros. Another helpful feature is **content assist,** which uses the information in a DTD or schema content model to provide a list of acceptable continuations depending on where the cursor is located in an XML file, or what has just been typed.

  The XML editor Source view also includes a "smart" double-clicking behavior. If your cursor is placed in an attribute value, one double-click selects that value, another double-click selects the attribute-value pair, and a third double-click selects the entire tag. This makes it easier to copy and paste commonly used pieces of XML.

- **Design view**: Represents the XML file simultaneously as a table and a tree. This makes navigation and editing easier for you. Content and attribute values can be edited directly in the table cells, while pop-up menus on the tree nodes

give alternatives that are valid for that location. For example, the **Add Child** menu item will list only those elements from a DTD or XML schema which would be valid children at a given point.

The Design view is especially helpful if you are new to XML, or need to do form-oriented editing. For example, you could use the Create XML File wizard to create a template XML file for a job description form from a job description DTD. After those steps are completed, you would only have to fill in the form data using the Design view.

Note: You can also use the **Outline** view to insert and delete elements and attributes.

3.8.2 DTD editor

The DTD editor, Figure 3-28, is a tool for creating and viewing DTDs.

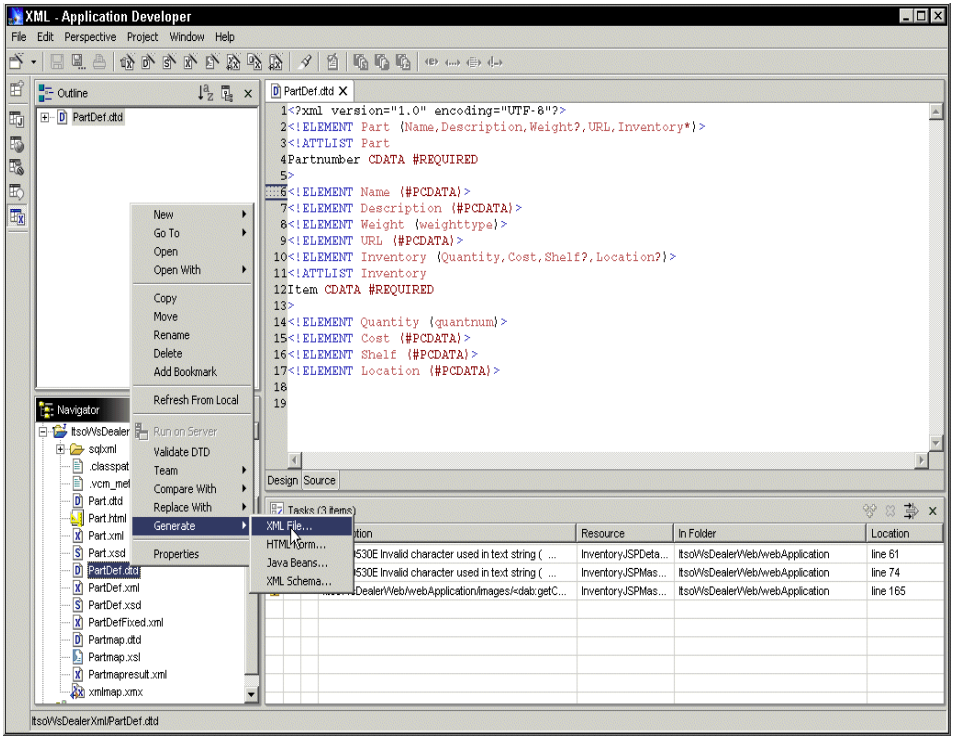


Figure 3-28 DTD editor

You can use it to perform a variety of tasks such as the following:

- Create new DTDs, either from scratch or from existing XML files.
- Validate and edit DTDs.
- Create and delete DTD elements, attributes, entities, notations and comments.
- Import existing DTDs for structured viewing.
- Generate an XML schema file from a DTD that you can further customize with the XML schema editor.
- Generate JavaBeans from a DTD.
- Generate an HTML form based on a DTD.
- Create an XML file from a DTD.

3.8.3 XML schema editor

XML schemas are an XML language for describing and constraining the content of XML files. XML schemas are currently in the Recommendation phase of the W3C development process.

XML schemas are a proposed alternative for the document type definitions, (DTDs), based on XML. They are a formal specification of element names that indicates which elements are allowed in an XML file, and in which combinations. A schema is functionally equivalent to a DTD, but is written in XML; a schema also provides for extended functionality such as data typing, inheritance, and presentation rules.

See `http://www.w3c.org/XML/Schema` for more information on XML schema.

Application Developer provides an XML schema editor for creating, viewing, and validating XML schemas.

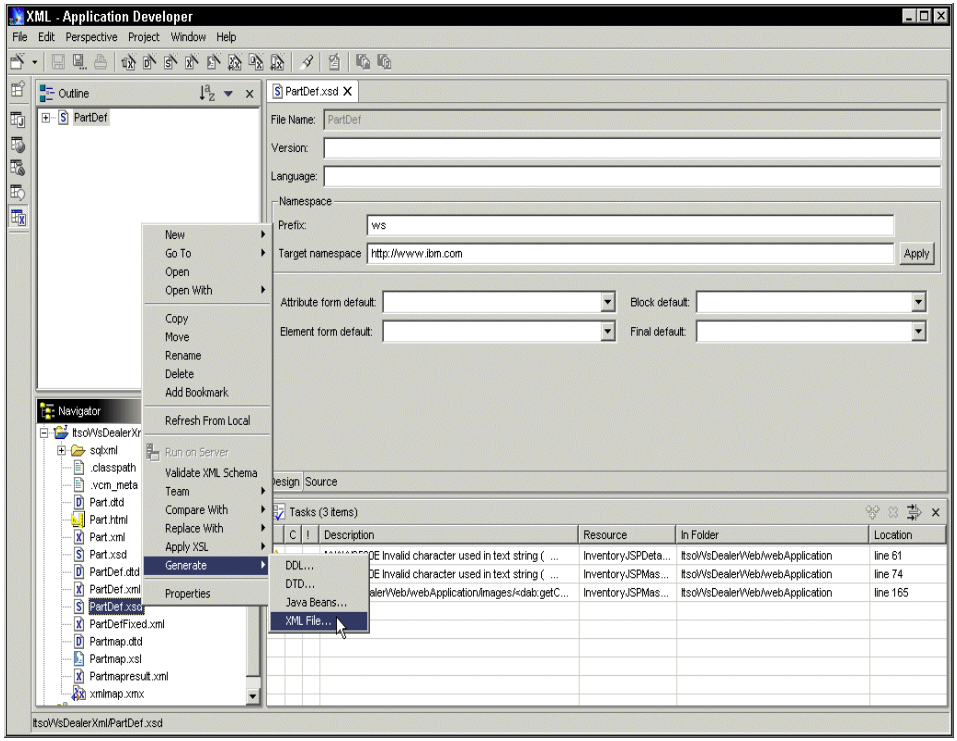


Figure 3-29 XML schema editor

Using the XML schema editor, you can do the following:

- Create and delete XML schema components such as complex types, simple types, elements, attributes, attribute groups, and groups.
- Edit XML schemas.
- Import existing XML schemas for structured viewing.
- Generate a DTD from the XML schema. You can then further customize that DTD with the DTD editor.
- Generate JavaBeans for creating XML instances of an XML schema.
- Generate XML files
- Generate relational table files.
- Validate XML schemas.

The XML schema support in the XML schema editor is based on the W3C XML Schema Recommendation Specification of May 2, 2001. The XML schema specification from the W3C Web site is used for validation.

3.8.4 XSL trace editor

The XSL trace editor is a kind of playback mechanism that is designed to work in conjunction with the XML to XML mapping editor. You can use the XML to XML mapping editor to generate an XSL style sheet by mapping information between a source and a target XML file. To test your generated XSL style sheet, you can use the XSL trace editor to apply the XSL style sheet to a source XML file and create a new HTML or XML file. You can then trace through the new XML or HTML file to verify if the results you received are correct.

The XSL trace editor records the transformation that is generated by the Xalan processor. The Xalan processor is an XSLT processor that transforms XML files into HTML, text, or other XML file types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath). Refer to the following Apache Web site `http://xml.apache.org/xalan-j` for more information on the Xalan processor.

The XSL trace editor enables you to visually step through an XSL transformation script, highlighting the transformation rules as they are fired. Additionally, an you can view the XML source and the corresponding XML or HTML result in "real time".

The XSL trace editor only works on a one-to-one basis. You can only apply an XSL file to one XML file (this file cannot reference others) at a time if you want to use the XSL trace editor to trace through the results. As well, you cannot apply an XSL file that references another XSL file to an XML file and then trace the results.

3.8.5 XML to XML mapping editor

The XML to XML mapping editor is a visual tool used to map one or more source XML documents to a single target XML document.

The tool accepts three types of input files:

- XML that includes either a `DOCTYPE` or an `xsi:schemaLocation` tag that indicates the corresponding DTD or XSD file respectively. If an instance document validates successfully with the XML editor, you can use it as input to the XML to XML mapping editor.
- An XML schema (XSD).
- A Document Type Definition (DTD).

The DTD or XSD contains the type information needed for generating the correct XSLT. An XML document alone does not have the necessary information for the tool to determine if an element is repeatable or non-repeatable, optional or required. The DTD or XSD provides the information needed for the transform to be successful.

If you do not have a DTD, you can use the DTD editor to generate a DTD from an XML document. With this generated DTD file, you can invoke the XML to XML mapping editor, or you can generate an XSD from the DTD. You may choose to use a DTD or XSD depending on whether or not you want to incorporate namespaces in your corresponding XML document.

You can use the XML to XML mapping editor to generate XSLT from HTML or XHTML if it is enclosed in a valid XML document. Although XHTML and HTML cannot be completely described by a DTD because they have generic tags, the DTD is used only for providing metadata and, therefore, does not have to be complete. As long as the DTD defines the portion of the document that you are interested in mapping, it is sufficient for the XML to XML mapping editor.

If you specify a DTD or XSD file, the tool will generate a sample XML document to be used by the XML to XML mapping editor. You can add XPath expressions, groupings, Java methods, beans, JavaScript, or conversion functions to your mapping. Mappings can be edited, deleted, or persisted for later use.

After defining the mappings you can generate an XSLT script. The generated script can then be used to combine and transform any XML documents that conform to the source DTD or XSD files.

The XML to XML mapping editor uses the Xalan processor to do transformations. Xalan is an XSLT processor that transforms XML documents into HTML, text, or other XML document types. The XSLT script that will be generated will be tested against the Xalan processor.

3.8.6 RDB to XML mapping editor

The relational database (RDB) to XML mapping editor is a visual tool used for mapping relational tables or an SQL statement to a DTD. After completing the mapping, you can generate a DAD file that can be run by the DB2 XML Extender to compose or decompose other XML documents into DB2 data.

There are 2 kinds of XML Collection DAD methods supported by the relational database to XML mapping editor, `RDB_node` and `SQL statement`.

- An RDB_node mapping DAD is created with a relational database table to a DTD mapping. The DAD can be used to store and retrieve data from DB2 databases.

- An SQL statement mapping DAD is created with an SQL statement (SELECT or FULLSELECT) to a DTD mapping. The DAD can be used to compose an XML document from an SQL query.

The mapping is stored in a session file with the extension `.rmx`.

Prerequisite: The RDB to XML mapping editor supports the IBM DB2 Universal Database. To deploy the generated DAD script, you must have IBM DB2, version 7.1 with Fix Pack 3, or higher.

To deploy the generated DAD script, you must have the **DB2 XML Extender** installed on your system. The DB2 XML Extender provides new data types that let you store XML documents in DB2 databases and new functions that assist you in working with these structured documents. Entire XML documents can be stored in DB2 databases as character data, or stored as external files but still managed by DB2. Retrieval functions allow you to retrieve either the entire XML document or individual elements or attributes.

See `http://www.ibm.com/software/data/db2/extenders/xmlext/library.html` for more information on the DB2 XML Extender, and select the latest documentation for your platform. Under the Administration section, open the XML Extender Administration and Programming guide.

3.9 Data perspective

You can access the Application Developer relational database tools from the Data perspective. The Data perspective lets you browse or import database schemas in the DB Explorer view, and create and work with database schemas in the Data view.

Editors and tools are provided to create and manipulate local descriptors, generate DDL from local descriptors, and run existing DDL to create local descriptors.

Local descriptors are kept in XMI files that can be manipulated with tailored editors in the Data view. These editors provide a graphical view of the data, editing is not performed on the XMI source file.

Through a connection in the DB Explorer view you can retrieve existing table definitions and convert the definitions into local XMI files.

XMI files and DDL files are used to store the definitions, and each format can be converted into the other; that is you can generate DDL from the XMI file, or generate the XMI file by executing the DDL file.

You can also generate an XML schema from a table definition.

All major database systems are supported.

The Data perspective provides views and tools for definition and maintenance of descriptors for database, schemas, and tables definitions. Multiple views are provided.

The Data perspective, Figure 3-30, contains four panes:

- **Top left**: Shows DB Explorer, Data and Navigator views.
- **Top middle**: Reserved for Editors (multiple files can be edited).
- **Top right**: Shows the Outline view (shows outline of the file in currently active editor).
- **Bottom**: Shows Tasks view (error messages).

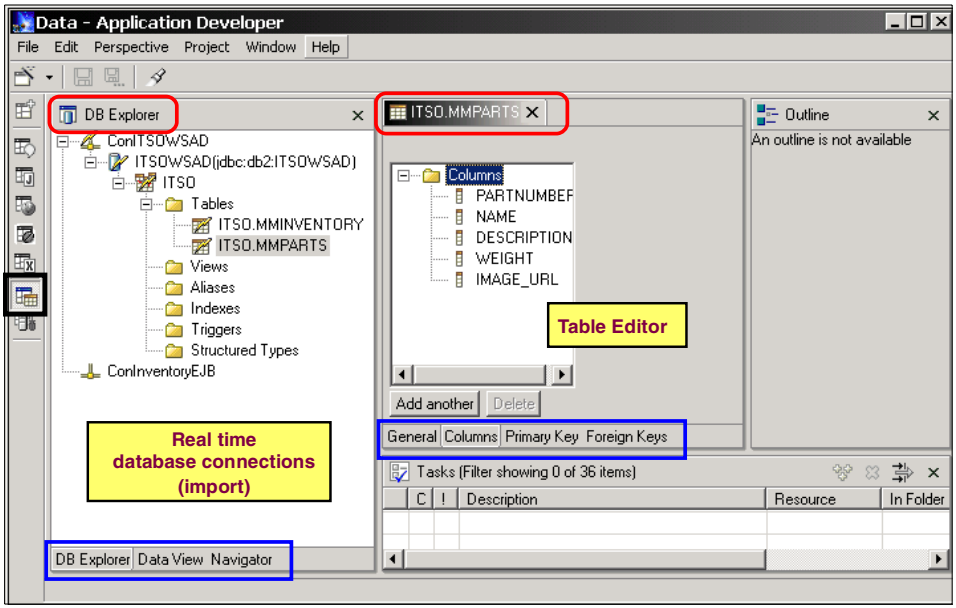


Figure 3-30 Data perspective

The Data view, Figure 3-31, shows database descriptors (XMI files) in a hierarchical format.

Editors are provided to define database, schema, and table descriptors. Tools are provided to generate SQL DDL files for such descriptors, or to create a descriptor from an existing DDL file. The DB Explorer view enables connections to database definitions in real time and import of existing descriptors as local resource files in XMI format.

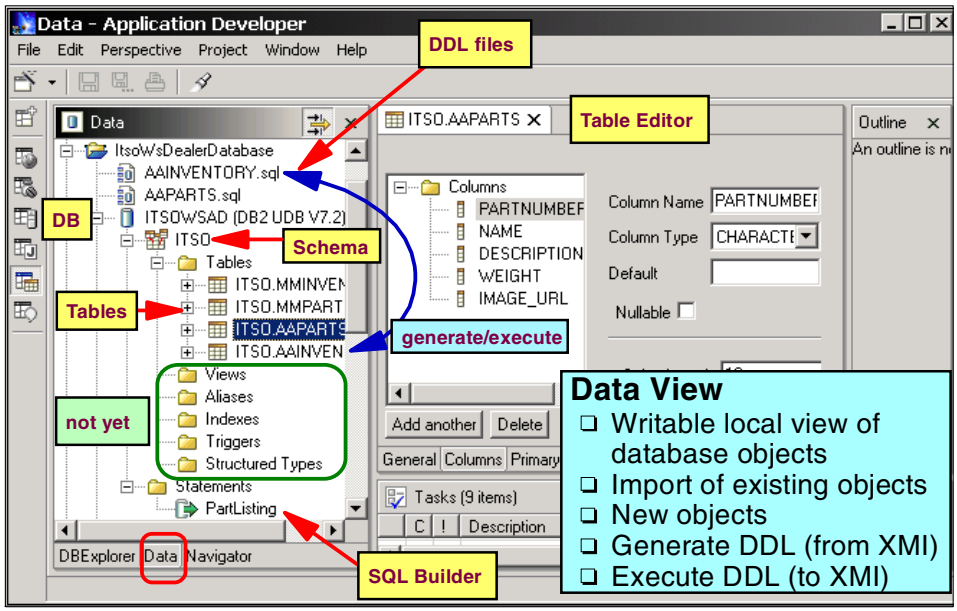


Figure 3-31 Data perspective - Data view

Through the DB Explorer view you can connect in real time to a database and retrieve information about the objects in it.

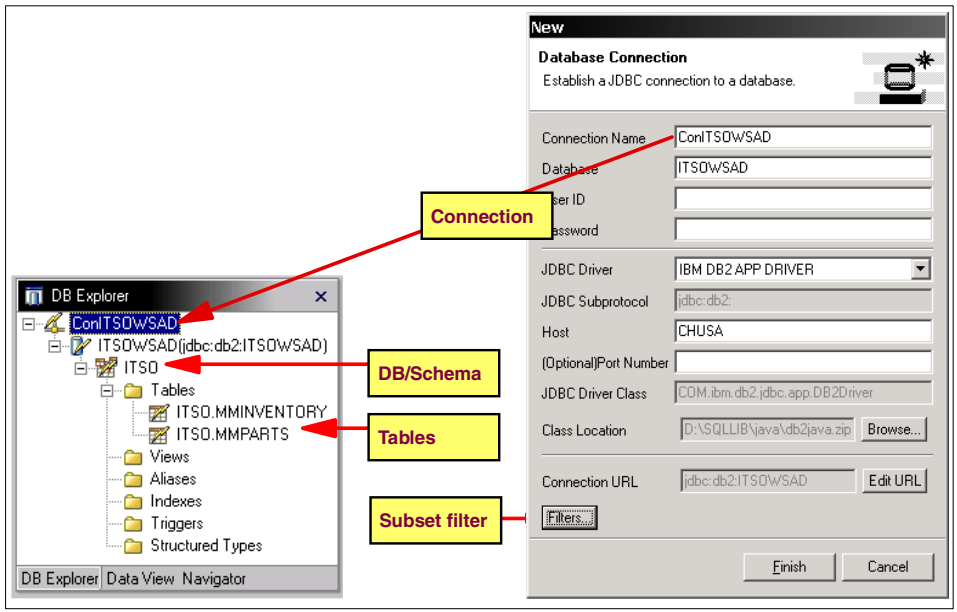


Figure 3-32 DB Explorer view and the database connection wizard

A wizard is provided to define the connection information, such as the JDBC driver. A filter can be setup to only retrieve schemas or tables that match a given pattern.

The DB Explorer is a read-only view. After viewing real-time information, the descriptors can be imported into a folder and then manipulated using the Data view. From the DB Explorer you can generate DDL files and XML schemas.

The Data perspective also provides a Navigator view, Figure 3-33, that can be used for operations on the underlying XMI files.

Invoking an editor on an XMI file from the Navigator view does, however, opens the same graphical editor as from the Data view.

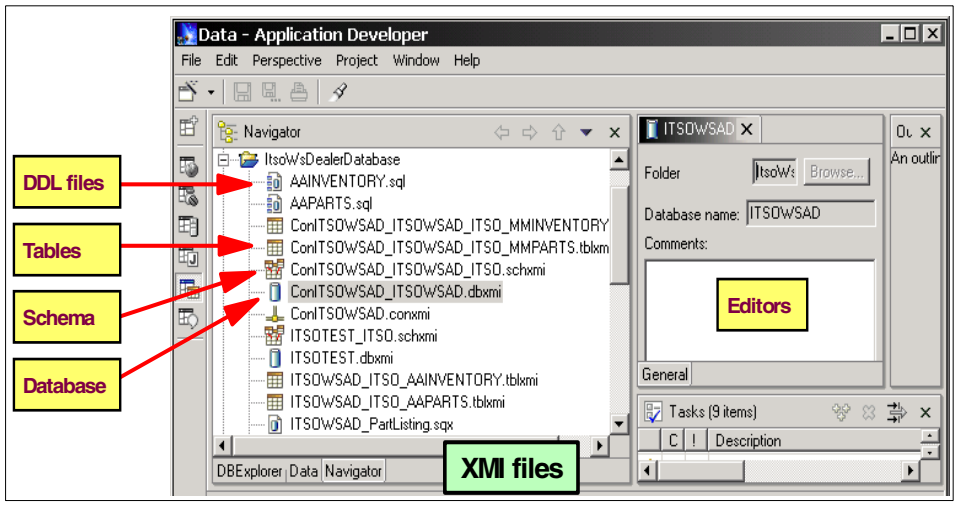


Figure 3-33 Data perspective - Navigator view

Wizards are provided to create databases, schemas, and table descriptors. For a table you can define the columns with their data types, as well as the primary key and foreign keys.

There are two tools available in Application Developer to create SQL statements:

- **SQL wizard**: Guided dialog that goes through a number of panels to select the table(s), select the columns, and to provide join, where clause condition, grouping and order information. SQL statements built with the wizard can be edited afterwards with the query builder.
- **SQL Query Builder**: Graphical editor to specify the tables, columns, join, conditions, group, and order. Instead of a guided dialog, a set of panels accessible through tabs is provided.

SQL statements can be built from an imported database model or through an active connection. You can run SQL statements against a real database, and you will be prompted for host variables that were defined in `WHERE` conditions.

SQL statements are stored as .sqx files, which are XML files.

Note: You can build `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements using these tools.

3.9.1 SQL Query Builder

SQL statements can be defined graphically using a wizard. Such statements can be executed for testing, and can be used by XML tools to convert database data into XML files. Statements can also be used by Web development tools to create skeleton Web applications (servlets and JSPs) that access databases.

The SQL query builder is provided by Application Developer to help you by building your database statements.

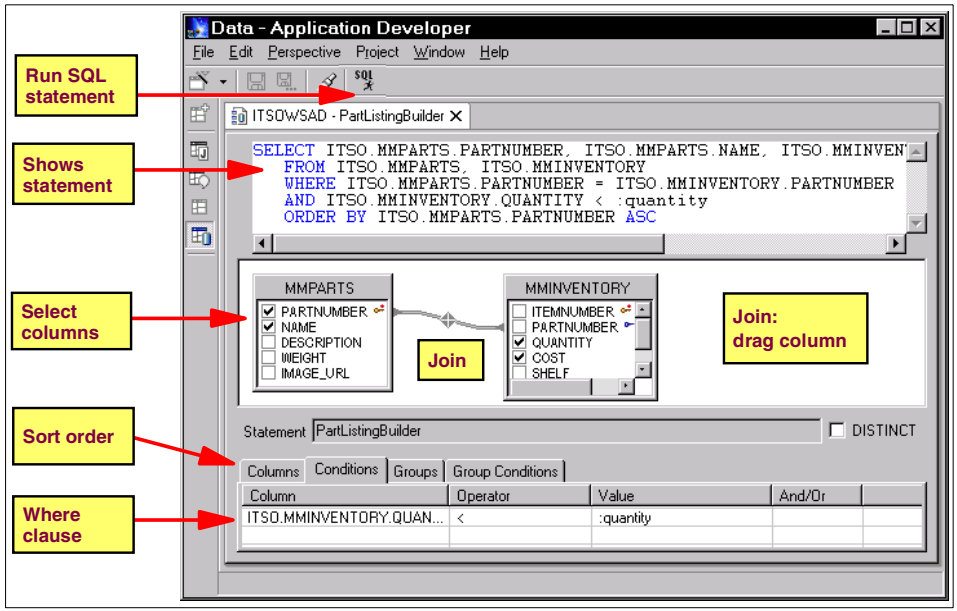


Figure 3-34 SQL Query Builder

The SQL query builder, Figure 3-34, is an editor with three panes:

- **Top**: Shows the actual SQL statement. You can edit the statement directly from here.
- **Middle**: Shows the tables, selected columns, and joins. You can drag and drop table objects from the Data view into this pane (and also into the top pane).

  A join is performed by dragging a column from one table to the matching column in another table.
- **Bottom**: Contains a set of panels that are accessible through tabs and are used to specify sort information, `WHERE` clause conditions, and grouping information.

An icon  is provided to run the SQL statement.

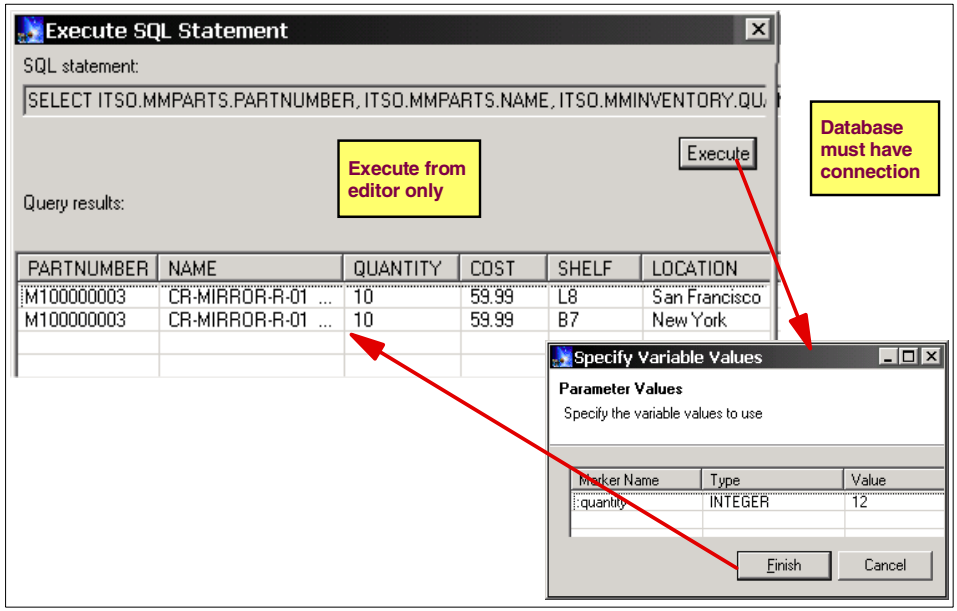


Figure 3-35 Executing an SQL statement from the SQL Query Builder

When an SQL query is executed you will be prompted for the values of any host variables used in `WHERE` clauses.

Results are displayed in table format for select statements.

SQL statements generated by the SQL query builder can be used by the XML tools to generate result data in XML format.

3.10 Debug perspective

Application Developer provides a Debug perspective that supports testing and debugging of your applications.

The Debug perspective, Figure 3-36, contains five panes:

- **Top left**: Shows Navigator, Processes and Debug views.
- **Top right**: Shows Breakpoints, Inspector, Variables and Display views.
- **Middle left**: Shows the Outline view of the currently displayed source.
- **Middle right**: Shows the Source view. This is an editor showing the line with the current error-/break- point, (where the process stopped).

- **Bottom**: Shows Tasks view (error messages) and the Console.

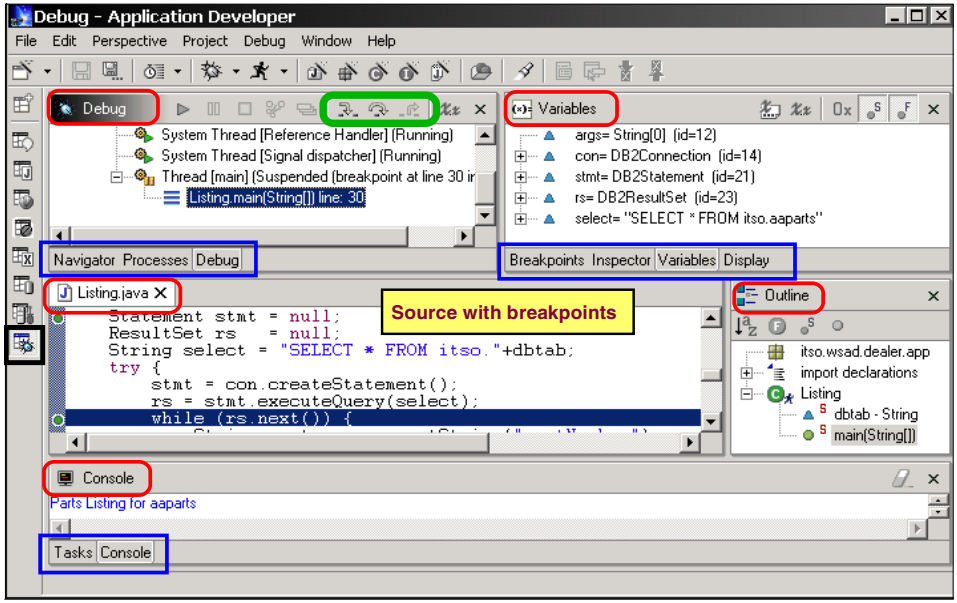


Figure 3-36 Debug perspective

The Debug perspective shows:

- **Processes view** with running processes.
- **Source view** showing the source of the Java program with the current line highlighted.
- **Outline view** showing the outline of the source code.
- **Console** showing the output.
- **Breakpoint view** showing all defined breakpoints.
- **Inspector** and **Variables views** showing with the current values.
- **Display view** where expressions can be evaluated and results are shown.

Tip: For easy debugging you may want to make the Variables view a separate window that you can move away from the Application Developer window. You can do so by simply dragging the view outside the workbench window and dropping it there. To return it back, you just need to do the opposite.

Debugging is supported on the local machine running Application Developer, or on a remote machine that has the *IBM Agent Controller* installed.

To debug Java code you set breakpoints in the source code. To set a breakpoint in the editor, double-click in the margin beside the line of code that you want to set as a breakpoint. An icon  will be shown to mark the line at which the breakpoint was set. A breakpoint set on a line of code will cause the execution of the program to stop at that line. You can also define a breakpoint to be activated only when a certain class of Java exception occurs. An exception breakpoint can be set by clicking the **Add Java Exception Breakpoint** icon  in the Breakpoints view.

To start a program in debug mode, click the **Debug** icon  . This will open the Debug perspective.

In the Debug perspective you use icons or shortcut keys to step into a line of code  , step over a line of code  , or to run to the end of a method  (Run to Return).

To remove a breakpoint in the editor, double-click the breakpoint icon  next to the line.

3.11 Profiling perspective

Profiling enables you to test your application's performance early in the application development cycle. This gives software architects enough time to make architectural changes with the knowledge that developers will also have sufficient time to implement those changes. This reduces risk early in the application development cycle, and avoids problems in the final performance tests.

The Profiling tools collect data related to the Java program's run-time behavior, and present this data in graphical and non-graphical views. This helps you to visualize your program execution easily, and explore different patterns within the program.

The tools are useful for performance analysis, and for gaining a deeper understanding of your Java programs. You can use them to view object creation and garbage collection, execution sequences, thread interaction, and object references. They also enable you to see which operations take the most time, and help you to find and solve memory leaks. You can easily identify repetitive execution behavior and eliminate redundancy, while focusing on the highlights of the execution.

Conventional performance tools, which are based on the procedural programming model, miss a lot of important information about the behavior of Java programs, which are object-oriented. The Profiling tools of Application Developer model and present your program's execution in a way that is natural and consistent with the object-oriented model, and that retains all relevant information.

The Profiling tools offered to you by Application Developer enable you to visualize the topology of your application, which is built from the following resources:

- Monitors
- Hosts
- Processes
- Agents

Profiling is described in more detail in 21.2, “Performance analysis” on page 459.

Besides being able to open this perspective from the main menu (**Perspective—>Open—>Other—>Profiling**), you can also open it from any of the other perspectives such as Java, Team, Scripts and so on (except Help), using the **Open Perspective** context menu for a project.

You can have multiple profiling sessions opened at the same time. To do so you just have to repeat one of the previously described ways of starting a new profiling session. Each new profiling session runs in a new Profiling perspective, and is identified by its own button on the vertical toolbar on the left border of Application Developer working area. You can switch between profiling sessions by clicking these buttons.

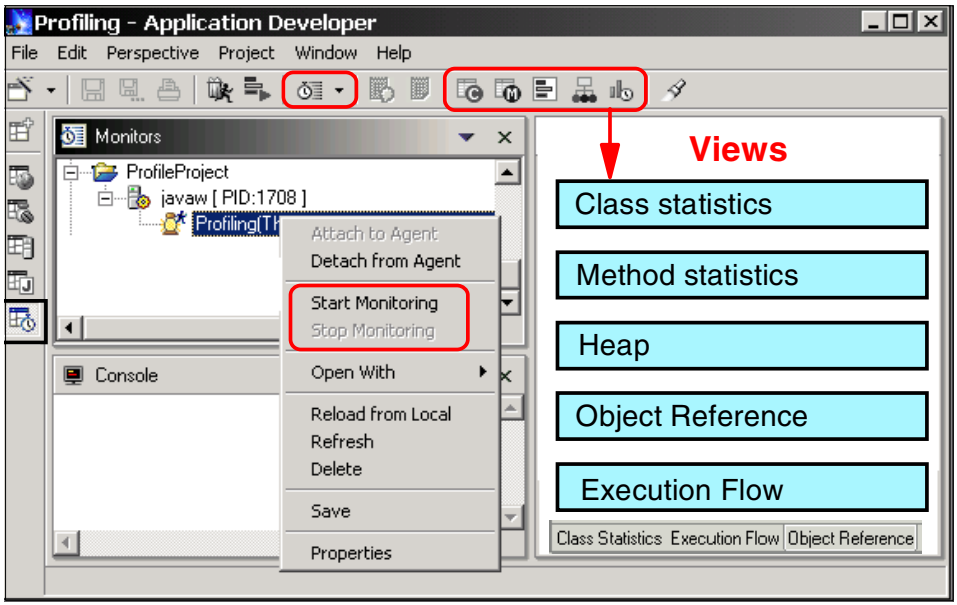


Figure 3-37 Profiling perspective

The Profiling perspective, Figure 3-37, contains three panes:

- **Top left**: Shows the Monitors view, which contains the profiling resources.
- **Bottom left**: Shows the Console view, which displays input, output and error data.
- **Right**: Shows the profiling views:
  - Heap
  - Execution flow
  - Object references
  - Method execution
  - Method invocation
  - Class statistics
  - Method statistics.

The individual views are used to display the following the performance data:

- **Class statistics** (tabular): Shows the time spent in each class.
- **Method statistics** (tabular): Shows the time spent in each method of each class.
- **Heap** (graphical): Shows the number of class instances.
- **Object references** (graphical): Shows all objects with their references to other objects.

- **Execution flow** (graphical): Shows a graphical view of the execution through the methods of the involved classes.

3.12 Script perspective

A Java developer uses the Java perspective to edit and build (compile) code, the Debug perspective to test the code, and the Script perspective to work with scrapbooks.

A Java scrapbook is a file with an extension of .jpage. It can be used to evaluate executable code fragments.You can enter any Java code in a scrapbook, select some or all of the code, and execute it.

You open a Script perspective the same way as any other perspective by selecting the **New Perspective** icon  in the left top corner of Application Developer working area, selecting **Other** from the menu item and selecting **Script** from the list of available perspectives offered in the **Select Perspective** dialog.

The Script perspective will be opened with no pages defined. To add a page to the Script perspective you click on the **Open New Wizard** icon  and select **Other**. Then select Java in the left list and Java Scrapbook Page in the right (Figure 3-38).

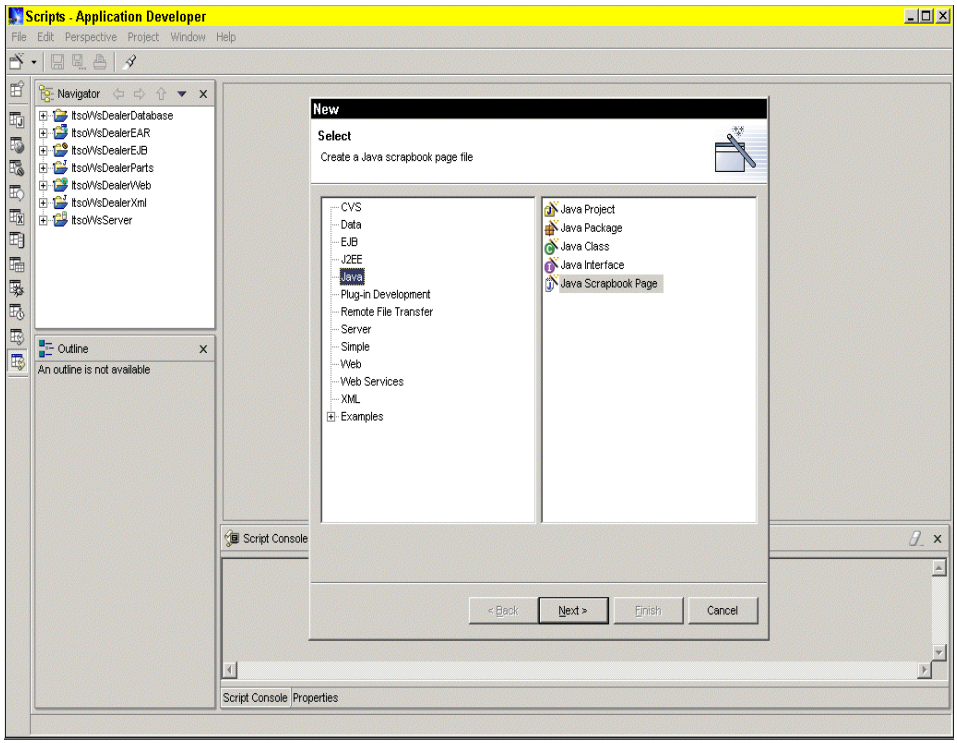


Figure 3-38 Adding Scrapbook page to Script perspective

Then click the **Next** button. A dialog will be displayed where you give the page a name and select the package context (Figure 3-39).

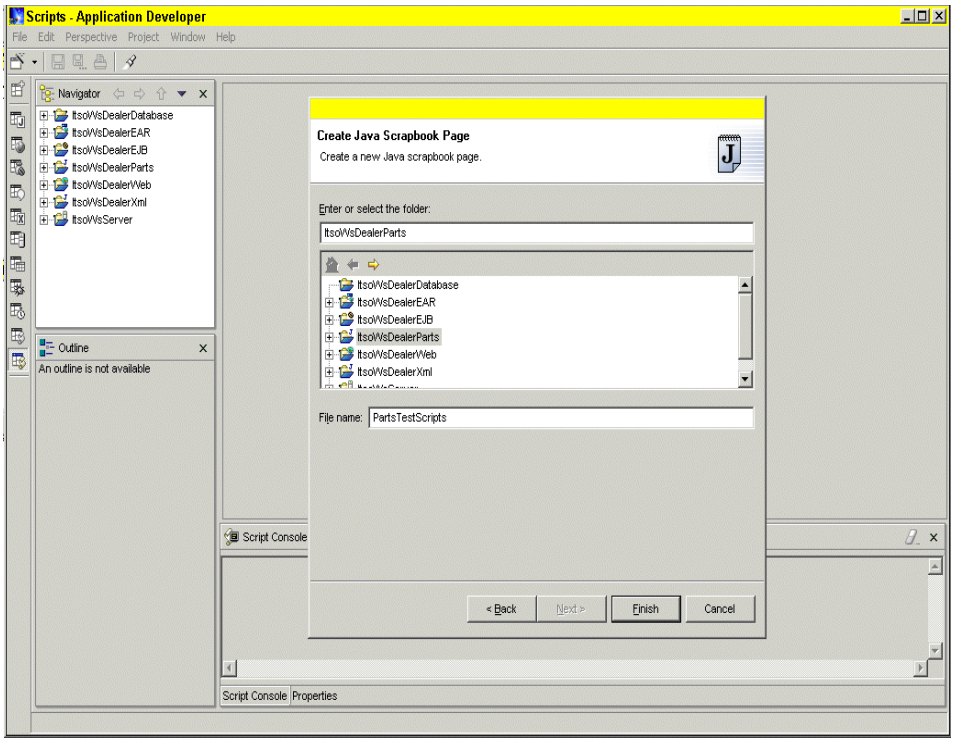


Figure 3-39 Setting the package context for Scrapbook page

The process of adding a Scrapbook page is completed by clicking the **Finish** button.

The newly created page is added in the Navigator view under the package you assigned it to. When you double-click it, its editor will be opened in the right pane.

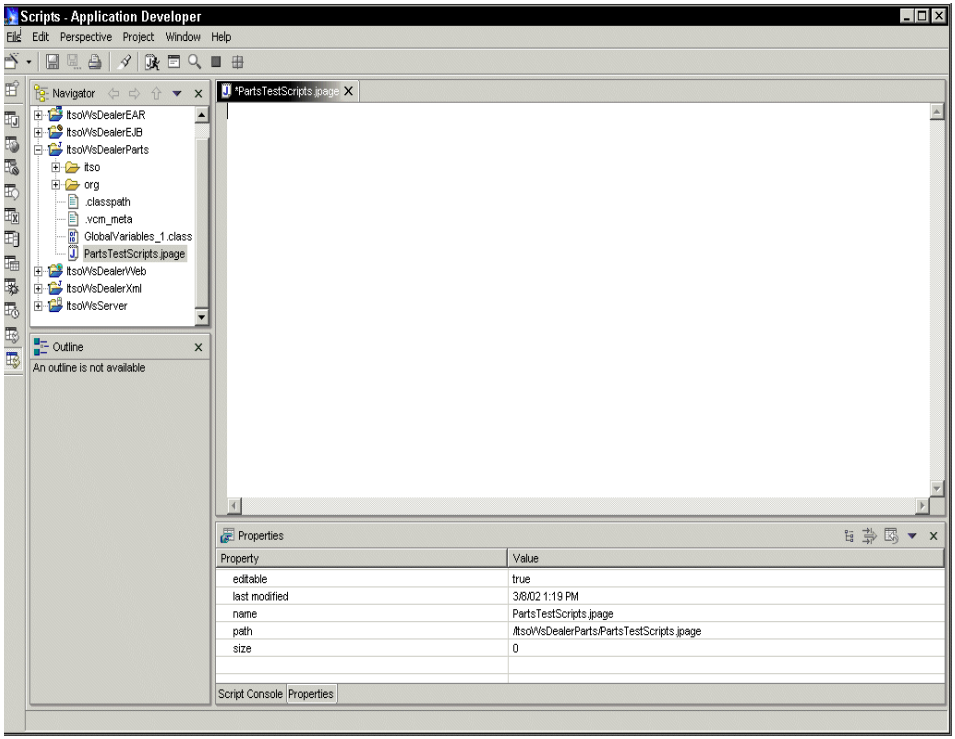


Figure 3-40 Script perspective with an open Scrapbook page

The Script perspective, Figure 3-40, contains four panes:

- **Top left**: Shows the Navigator view, which provides a hierarchical view of the resources in the workbench.
- **Bottom left**: Reserved for the Scrapbook pages.
- **Top right**: Shows the Source view.
- **Bottom right**: Shows the Console and Properties views, which display property names and values of a selected resource.

You can use the Scrapbook page to experiment and evaluate Java expressions, (code snippets). Snippets are edited and evaluated in the Scrapbook page editor. In the editor you can select a code snippet, evaluate it, and display the result as a string or show the result object in the debugger's inspector.

3.13 Team perspective

Application Developer maintains a workspace where the project data is stored. By default it is the directory:

```
C:\<Application Developer_ROOT>\workspace
```

The workspace directory can be specified when the Application Developer is started (`-data workspacedirectory`).

The interactive development environment has the following facilities:

- All deletes are permanent. There is no “recycle bin”.
- A history of all changes is maintained locally and files can be reset to a previous state.

To enable the multiuser development environment, a shared repository and a version and configuration management system are required. These are used to store, maintain and share development resources.

A repository is a persistent store that coordinates multi-user access to the resources being developed by a team.

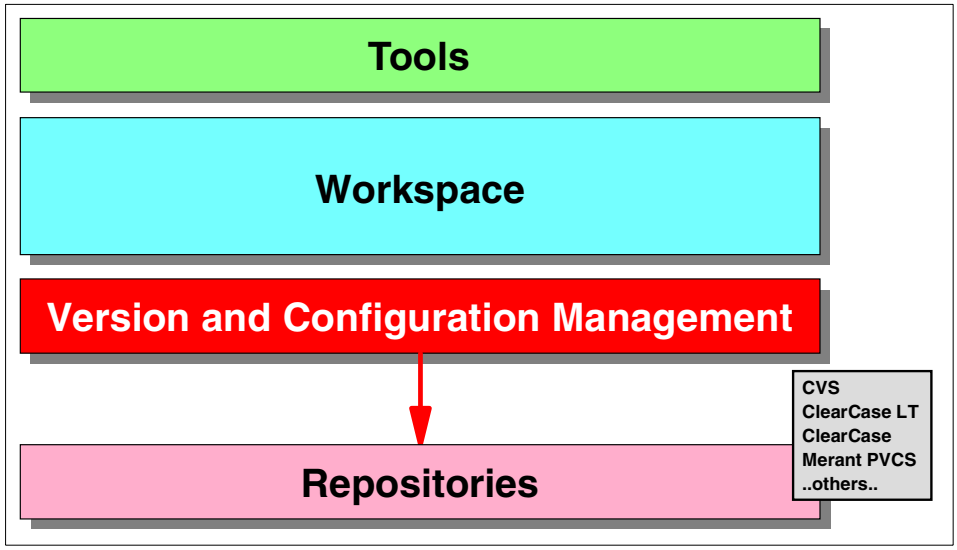


Figure 3-41 Version and configuration architecture of the workbench

The version and configuration management architecture of the workbench platform, Figure 3-41, enables different vendor tools to be enabled on it. Plug-ins for two of them, (Rational ClearCase LT and Concurrent Versions System - CVS), are included in the base Application Developer product. Other vendors can provide plug-ins for their tools.

Application Developer uses optimistic locking to solve sharing conflicts. In the optimistic concurrency model, any developer can change any code. This is based on the assumption that conflicts are rare because developers usually work on different files or if they work on the same file they do so in a “sequential” order, which means another developer starts modifying the resource only after changes of the first one have been released.

However, conflicts where different developers change the same file at the same time do occur and must be dealt with.

The Team perspective, Figure 3-42, is used to manage projects in conjunction with a shared repository.

The Team perspective contains four panes:

- **Top left**: Shows Navigator view.
- **Top right**: Shows Repositories and Synchronize views.
- **Bottom left**: Shows Properties view.
- **Bottom right**: Shows Tasks view and Resource History view.

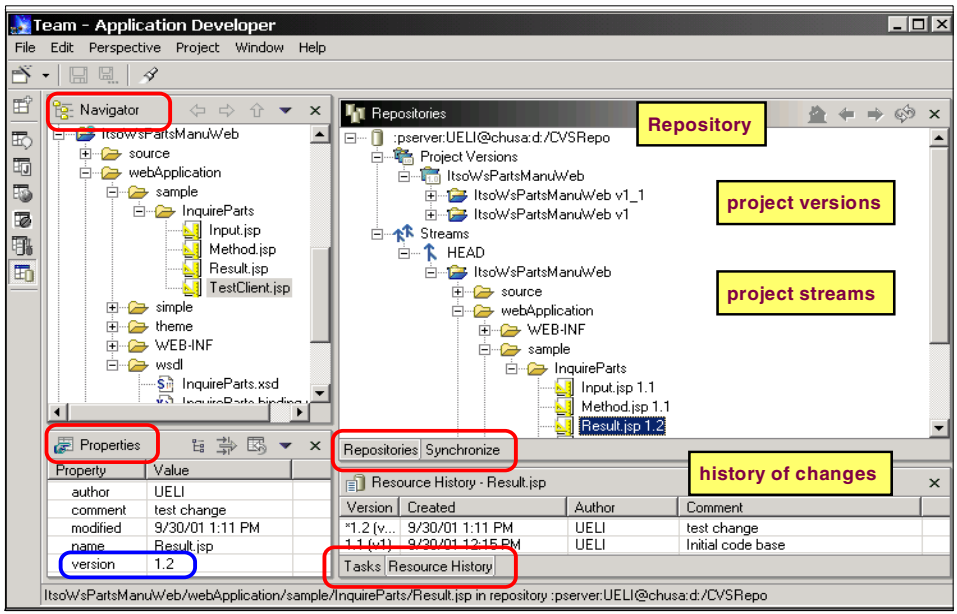


Figure 3-42 Team perspective

The Repositories view displays the repository connections, the project versions, and the active project streams within the projects.

The Synchronize view displays the changes between files in the local workspace and the team stream.

The Resource History view shows the sequence of changes performed on one file.

The first task to do is to define a new or connect to an existing shared repository.

Next projects from the local workspace are added into the team stream by invoking the **Release** action. This makes projects available in the shared environment for other team developers. An initial version of the project can be established as well.

Team members can then access projects from the repository by selecting the project from the team stream and adding it to their workspaces.

The actions that the individual team members can perform from now on are:

- **Compare** the project in the workspace with the project in the team stream.
- **Replace** the project in the workspace with the version in the repository.

- **Show the history** of a file, that is the changes of all developers that touched the file.
- **Synchronize** the project in the workspace with the team stream.

  A dialog showing all the differences will be displayed. From this list, the team member can decide to:

  - **Release** changes to the team stream
  - **Catch-up** changes of other developers into the workspace.
- **Resolve conflicts** which are displayed if the same file has been changed by the team member and other developers. Conflicts must be resolved by merging the changes.
- **Version** the project from the workspace or from the team stream.

Tip: To enable versioning of project data we suggest that you use a versioning system even when working in a single workstation environment.

3.14 Help perspective

The workbench provides an online help system that allows the developers to provide online help for their application and tools. It lets the user access the entire documentation, and allows browsing, searching and printing.

3.14.1 Workbench Help tools

The documentation in the help system is organized into *information sets* and *information views*. These, along with a full-text search engine and context-sensitive interface help, are designed to help the user find any necessary information quickly and easily.

An information set is a group of online help topics that make up the online documentation for a discrete subject area, usually for a whole product or for a major part of it. Essentially, it is the documentation for the whole product.

Sometimes the documentation for a product is so large so that it needs to be divided into several information sets, with a different focus for each. In a product with more than one information set, the user can select the one to view from the drop-down list above the navigation frame. Application Developer itself is an example of a product with several information sets.

An information view is a navigation tree, presented on the left-hand side of the Help view, which lets the user navigate through the topics in the information set.

Every information set has one or more information views. If there are multiple information views provided with an information set, the user can switch between them using the tabs at the bottom of the navigation frame. Each information view presents links to the same topics as the other views, but ordered in a different way.

If the user wants to find a particular piece of information in the online help, a Search function is available. The help system will scan the current information set, or only a part of it, and return a list of topics that meet the specified search criteria.

If the user is stepping through a task and encounters something that requires clarification, Application Developer provides a context sensitive help feature called an *infopop*. This feature is activated by setting focus to the interface widget in question, either by clicking on it or using the Tab key, and then pressing **F1**. The infopop will provide information about the widget and, if appropriate, show links to where more information can be found.

From the Help view the user can select to print out a topic or a group of topics, so that it can be read off-line or marked up for reference.

3.14.2 Application Developer Online Help

Application Developer uses the workbench help tools to provide its own online help.

The help system is presented in the Help perspective, Figure 3-43, which behaves like other perspectives in the workbench.

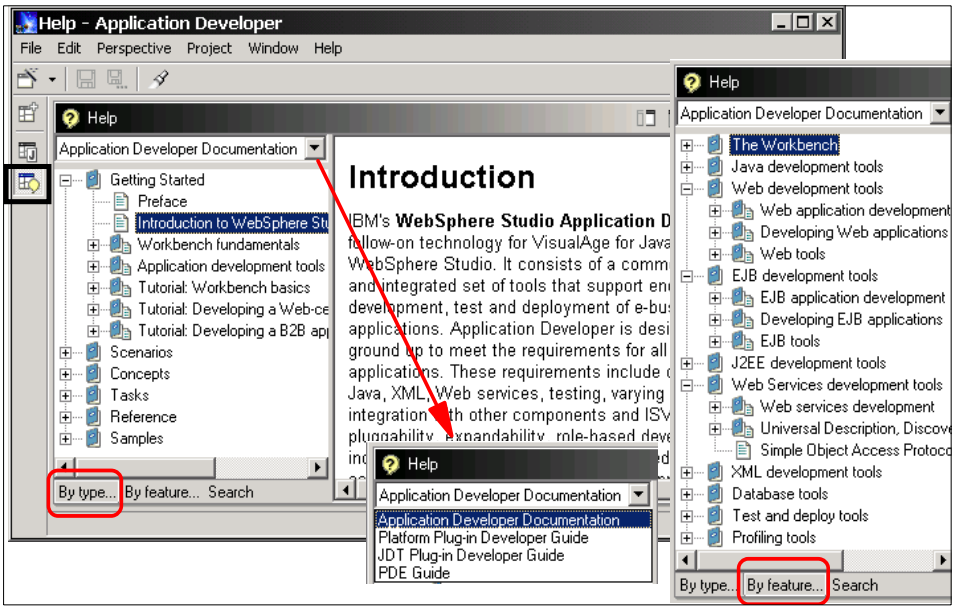


Figure 3-43 Help perspective

Documentation can be accessed **By type**, shown at left in Figure 3-43, or **By feature** shown at right in Figure 3-43. The help facility also covers the underlying workbench with a platform plug-in developer guide, a JDT (Java Development Tools) guide, and a PDE (Plug-in Development) guide.

As an alternative to opening a Help perspective, a Help view can be added to any other perspective by performing the following steps:

1. Select **Perspective—>Show View—>Other**.
2. Expand the **Help** node in the tree, and select **Help**.
3. Then click **OK**.

This adds the Help view to the current perspective. Usually the view that is added is too small to display the help very well. To change the size of the Help view you can:

1. Drag the title bar of the Help view to the shortcut bar (on the left-hand side of the workbench). This adds a button for the Help view to the shortcut bar.
2. Click the button to open the Help fast view.
3. Adjust its right-hand edge until it is a usable size.

To close the fast view, click its button on the shortcut bar again.

To navigate the online help (Figure 3-43):

1. Use the drop-down list in the Help view, (immediately above the navigation frame), to select the information set that you want to browse. The drop-down list appears only if there is more than one information set available.
2. Switch navigation views by clicking the tabs at the bottom of the navigation frame. Each view contains links to topics in the current information set, organized in different order.
3. Expand the topic tree to find the information you are looking for. To view a topic, click the link in the topic tree.

Most topics provide a list of links to related topics at the bottom. Follow these links to learn more.

You can use the **Forward** and **Back** buttons in the Help view toolbar. These behave the same way back and forward buttons work in an Internet browser, taking you to topics you have already looked at.

To synchronize the navigation frame with the current topic you can click the **Synchronize** button. This is helpful if you have followed several links to related topics in several files, and want to see where the current topic fits into the navigation path.

As an alternative to browsing the information this way, you can use the search engine. To do so click the **Search** tab at the bottom of the navigation frame. The search engine searches only the current information set.

To see context-sensitive help from the workbench set focus to a particular widget and press **F1**.

3.15 Tasks view

Application Developer provides you with several types of markers including bookmarks, task markers, debugging breakpoints and problems. In this chapter we will focus on tasks and the Tasks view.

The Tasks view displays all the tasks and problems in the workbench, both those associated with specific files as well as generic tasks that are not associated with any specific file.

The Tasks view displays the following information:

- Auto-generated errors, warnings, or information associated with a resource. These are typically produced by builders.

- Tasks that have been added by the user. A task may be associated with a resource, or it may be global.

Figure 3-44 shows the Tasks view.

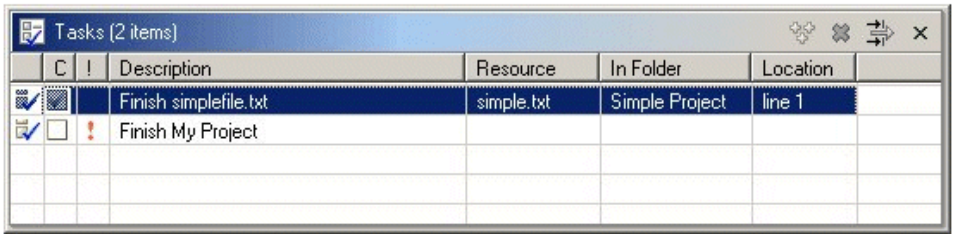


Figure 3-44 Tasks view

The following information is shown in the columns of the Tasks view:

- **Type**: Displays an icon denoting the type of task. This can be one of: Task, Error, Warning, or Info.
- **Completed**: Indicates whether the task is completed or not. A task is completed if a check mark is indicated. You can manually mark a task as completed.
- **Priority**: Indicates whether the task is of high, normal, or low priority. You can change the priority of a task from the combo box in this column.
- **Description**: Contains a description of the task. You can edit the description of user defined tasks by clicking in this column.
- **Resource**: Contains the name of the resource with which the task is associated. For global tasks, this column is blank.
- **In Folder**: Indicates the folder containing the resource with which the task is associated. For global tasks, this column is blank.
- **Location**: Indicates the line number in the associated file where the task marker is located, if there is an associated file.

The following commands can be performed from the Tasks view:

- **New Task**: Creates a global task that is not associated with a resource.
- **Delete**: Deletes the selected task or tasks.
- **Filter**: Brings up a dialog which allows you to filter the display of tasks according to the type of task.

The Figure 3-45 shows the Filter Tasks dialog.

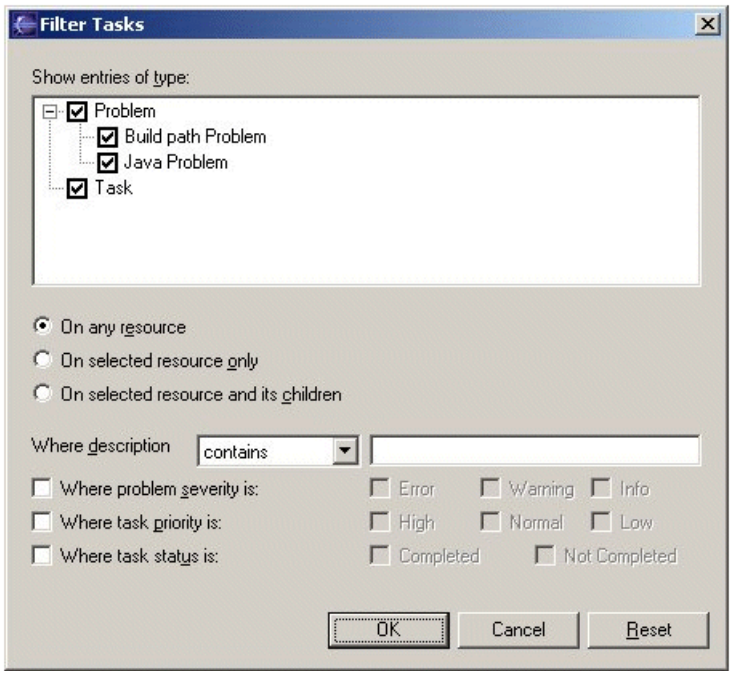


Figure 3-45 Filter tasks dialog

The selections in this dialog control what tasks are displayed in the Task list:

- **Show Entries of Type**: Displays in the Tasks list of the Task view only the elements selected in the list.
- **On any resource**: Shows tasks associated with all resources.
- **On selected resource only**: Shows only tasks associated with the selected resource.
- **On selected resource and its children**: Shows only the tasks associated with the selected resource and its children.
  - **Where description** - Specifies a string to match the description against:
  - **contains**: Shows only the tasks whose descriptions contain the specified string.
  - **does not contain**: Shows only the tasks whose descriptions do not contain the specified string.
- **Where problem severity is** - Can be used to indicate what kind of problem tasks to display. You may choose one or more of the following options:
  - Error
  - Warning

- Info

Note: If "Problem" was not selected in the "Show entries of type list", then this option will not be available.

- **Where task priority is** - Can be used to indicate what priority level of tasks to display. You may choose one or more of the following options:
  - High
  - Normal
  - Low

Note: If "Task" was not selected in the "Show entries of type" list, then this option will not be available.

- **Where task status is** - Can be used to indicate what status of tasks to display. You may choose one or more of the following options:
  - Completed
  - Not Completed

Note: If "Task" was not selected in the "Show entries of type" list, then this option will not be available.

4

Application Developer terminology

This chapter introduces some of the main terms used in Application Developer.

- J2EE architecture
  - EAR files
  - WAR files
  - JAR files
- Projects and folders
- Project types
  - Simple
  - Java
  - Web
  - Enterprise application
  - EJB
  - Application client
  - Server
- Project organization

© Copyright IBM Corp. 2002

4.1 J2EE architecture

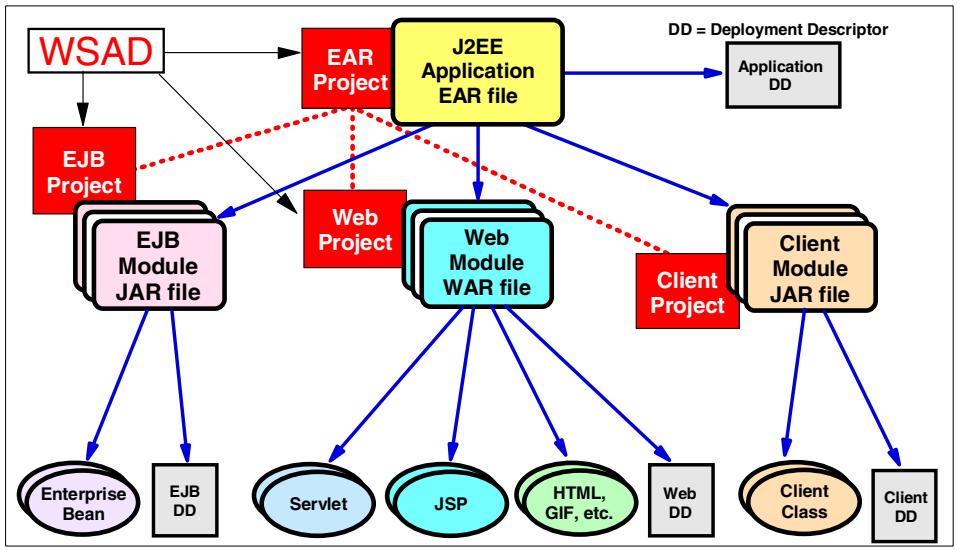


Figure 4-1 J2EE architecture

Figure 4-1 shows the J2EE hierarchy and the matching support in the Application Developer.

A J2EE application is stored in an Enterprise Archive (EAR) file, that contains EJB modules (stored in an EJB JAR file), Web modules (stored in Web Archives (WAR) files), and client modules (stored in a JAR file).

Each of the modules contains a deployment descriptor, for example, a WAR file contains a web.xml file.

A WAR file contains all the components of a Web application, that is, servlets, JSPs, HTML files, images, and so forth.

The J2EE hierarchy is matched by projects in Application Developer. An EAR project contains references to EJB, Web, and client projects. A Web project contains all the resources (servlet, JSP, HTML, images) and the deployment file (web.xml).

This setup makes deployment to a J2EE-based application server very easy.

4.2 Projects

Application Developer organizes all resources (data, programs) into projects.

A *project* is the top level of organization of resources in the workbench. A project contains files and *folders*. Projects are used for building, version management, sharing, and organizing resources. A project can contain session and persistent properties, settings for environmental variables, and references to other projects.

Builders are used to create or modify resources within projects, usually based on the existence and state of other resources. Builders enforce the constraints of a domain. For example, a Java builder converts Java source files into executable class files, a Web link builder updates links to files when names and locations have changed, and so on.

As resources are created and modified within a project or projects, builders are run and constraints are maintained. Builders associated with resources perform the necessary actions to ensure the resources are executable.

A project is either *open* or *closed*. When a project is closed, it cannot be changed in the workbench, it cannot be referenced from the other projects. The resources of a closed project will not appear in the workbench, but the resources still reside on the local file system.

Tip: Closed projects require less memory. Since they are not examined during builds, closing a project can improve build time. If your project is completed and does not have to be modified, but needs to be used with the other project, we recommend that you pack it into a JAR file and add JAR as an external JAR file in the classpath, then close your project to avoid compile errors.

When a project is open, the structure of the project can be changed and you can see the content of the project. When working with one of the J2EE project types (Web, Enterprise Application, EJB, application client), all projects associated with the enterprise application must be open.

A project is a grouping of resources needed for a specific task. There are unique project types for developing J2EE applications:

- **Simple project:** Generic projects that contain files and folders.
- **Java project**: Contains the resources needed for Java applications, including Java files and class files. Java projects are used to create Java packages.

  When you create a new Java project, the environment is set up for Java development. A Java builder is associated with the project so the Java source can be incrementally compiled as it is updated.

The Java project contains information about the type hierarchy and Java elements. This information is kept current as changes are made.

Web, EJB, and application client projects can also contain the resources needed for Java applications. The Java builder will incrementally compile the resources within these projects as the resources are updated.

The Java packages can be exported as Java archive (**JAR**) files and included in the build path for other project types, or in corresponding application class path for the application servers.

Each Java project maintains additional information about the type hierarchy and the references and declarations of Java elements. This information is constantly updated as the user changes the Java source code; it is not dependent on the builder.

- **Web project**: Contains the resources needed for Web applications, including servlets, JSP files, Java files, static documents (for example HTML pages or images), and any associated metadata.

  Web applications are packaged and deployed as Web archive (**WAR**) files. WAR files are used to import Web application into Web servers.

  A Web project reflects the structure and hierarchy of folders and files necessary to permit Web module-level activities to be performed by various roles.

  When you create a Web project, you must specify an Enterprise Application project for the Web project to belong to. Web projects are always imbedded in Enterprise Application projects and a Web module element is added to the application.xml deployment descriptor for the Enterprise Application project.

  Web modules are also known as Web applications, in servlet programming.

- **Enterprise Application project:** Contains the resources needed for enterprise applications and can contain a combination of Web modules, EJB modules, JAR files, and application client modules.

  Enterprise Application projects are exported as Enterprise archive (**EAR**) files that include all files defined in the Enterprise Application project as well as the appropriate module archive file for each J2EE module project defined in the deployment descriptor, such as Web archive (WAR) files and EJB JAR files.

  In Application Developer, the modules in an Enterprise Application project are mapped to other J2EE projects. The mapping information is stored in metadata files within the Enterprise Application project. The metadata files are used for exporting the project to an EAR file, and for running the project on the server.

  If you do not specify an Enterprise Application project when you create a J2EE project, one will be automatically created for you.

- **EJB project**: Contains the resources for EJB applications. The EJB project contains the metadata files (such as the deployment descriptor, IBM extensions, and rdb mappings) for the EJB application, Java source files, compiled code for the enterprise beans, and stubs for the beans.

  The source files for your EJB project will be kept in the EJBModule folder and the binary files will be kept in the bin folder of the EJB project.

  An EJB project is deployed as an EJB module - *JAR file*.

- **Application client project:** Contains the resources needed for application client modules. Application client projects are typically run on networked client systems connected to J2EE (EJB) servers.

  The source files for your application client project are kept in the appClientModule folder and the binary files are kept in the bin folder of the application client project.

  An application client project is deployed as a JAR file.

  This application client JAR file contains all of the class files that a client program needs to use the enterprise beans that are contained in an EJB module.

  A J2EE application client container provides access to the J2EE service (JNDI naming services, deployment services, transaction services, and security services) and communications APIs (Internet protocols, Remote Method Invocation protocols, Object Management Group protocols, Messaging protocols, and data formats).

  Like Java projects, application client projects contain the resources needed for application clients, including Java files and class files.

  When you create a new application client project, the environment is set up for Java development. A Java builder is associated with the project so the Java source can be incrementally compiled as it is updated.

  The application client project contains information about the type hierarchy and Java elements. This information is kept current as changes are made, and the Java builder will incrementally compile the resources within these projects as the resources are updated.

  An application client project enables you to do the following things:

  – Develop the Java classes that implement the client EJB module
  – Set the application client deployment descriptor
  – Test the application client

  In Application Developer, application client projects are always embedded in Enterprise Application projects. When you create an application client project, you must specify the Enterprise Application project to which the application client project belongs.

An appClient module element is automatically added to the application.xml deployment descriptor for the Enterprise Application project.

- **Server project:** Describes a test environment for applications. Server projects contain the information necessary to deploy an application to an application server for testing.
  - The Server perspective of Application Developer shows server instances and server configurations you have defined to test your projects.
  - Server instances identify servers where you can test your projects.
  - Server configurations contain setup information.

Since projects are the root of any application development, you will always start developing a new application by creating one or more projects.

Application Developer provides wizards to create each specific type of project. All the wizards for creating projects are similar, with slight variations required for the project type.

In general, when you create a new project you specify the following:

- The name of the project
- The location to use
- File organization
- Dependencies between modules
- Build output folder
- Projects and libraries that need to be in the build classpath (if module dependencies have not been specified).

By default project and their folders are created and stored in a workspace directory (`./Application Developer_Installdir/worskspace`), where all the metadata is stored.

In the team programming environment of WebSphere Studio, projects can be associated with a team stream and they can be versioned.

Each project has a type; Java projects are for stand-alone applications, Web projects for Web applications, EJB projects for EJB development, EAR projects tie together Web and EJB (and Client) projects into a J2EE hierarchy, Server projects are used to define application servers for testing.

Perspectives are the way a developer sees the projects. Perspectives are tailored for certain tasks, based on the role of the developer. As we have seen already, perspective is a set of views, arranged into the workbench window, and a set of editor and tools that are used to manipulate these resources. Perspectives can be customized to better match a user's role.

In most development efforts you already have existing resources. These resources can be imported into Application Developer projects in many ways:

- Files from directories (Java source, HTML, JSPs, ...)
- ZIP and JAR files (import as individual files for editing, or leave as ZIP/JAR file if only used for compilation)
- EJB JAR files with existing EJB definitions (for example, from VisualAge for Java)
- EAR and WAR files from existing Web applications and J2EE archives
- FTP or HTTP access to existing Web sites for import of HTML and associated files

Application Developer provides a comprehensive validation for many types of projects, for example, the EJB validator checks whether the EJB 1.1 specifications are followed in the Java code.

Projects can be validated automatically when a resource is saved (this is the default) or validation can be performed manually on demand.

Projects are in Application Developer associated with servers. You can assign a preferred (default) server for each project. When a server is started the associated projects are loaded and their code can be executed.

4.3 Project organization

We will now look at how the different types of projects are organized.

4.3.1 Enterprise project organization

To start developing J2EE applications, you first need to create an Enterprise Application project (called an Enterprise Application Module) to contain your Web, EJB, and application client modules.

The Enterprise Application project is used to collect an entire application from the various modules.

4.3.2 Java project organization

You can organize Java projects in two different ways:

- Using the project as the source container.

  In this organization, all Java packages are created directly inside the project. This is the recommended organization for simple projects.

  The generated .class files are stored along with the .java source files.

- Using source folders as the source container.

  In this organization, packages are created not directly inside the project, but in source folders. You create source folders as children of the project and create your packages inside these source folders.

  This is the recommended organization for more complex projects. It allows you to subdivide packages into groups.

Depending on the file organization you choose when creating the project, the class files are either stored along with the Java files in the project folder or, if you are using source folders, the class files are stored in a separate output folder.

The builder will also copy non-Java resources (for example, GIF files) into the output folder. If you have non-Java resources that should not be copied into the output folder, for example a readme file, you can create an ordinary folder and store the resources there.

4.3.3 Web project organization

A WAR file has a specific hierarchical directory structure. The default directory structure in the Web project adheres to the J2EE specification.

This specification defines a project directory structure that specifies the location of Web content files, class files, class paths, deployment descriptors, and supporting metadata.

The Web project hierarchy mirrors that of the Web application created from a project.

The webApplication folder contains all of your Web resources, including HTML, JSP, graphic files, cascading style sheet, web.xml (deployment descriptor), classes, and lib folders (not served directly to a client).

- The top level directory is the **document root** of the application.
- The document root is where JSP pages, client-side classes and archives, and static Web resources are stored.

- The document root contains a subdirectory called WEB-INF, that contains the web.xml deployment descriptor, tag library descriptor files, classes, and lib directories.

The Web project uses a J2EE directory structure in organizing the following folders and files:

- **source**: Contains the project Java source code for classes, beans and servlets.

  When resources are added to a Web project, they are automatically compiled and the generated files are added to the webApplication/WEB-INF/classes directory.

  By default the content of the source directory is not packaged in WAR files.

- **webApplication**: Holds the content of the WAR file that will be deployed to the server.

  It contains all the Web resources, including HTML files, JSP files, and graphics needed for the application.

  Any files not under webApplication are considered development resources (for example `.java` and `.sql` files) and will not be deployed when the project is unit tested or published.

- **webApplication/theme**: Contains cascading style sheets and other style-related objects.

- **webApplication/WEB-INF**: Contains the supporting Web resources for a Web application, including the web.xml deployment descriptor file and the classes and lib directories.

- **webApplication/WEB-INF/classes**: Contains servlets, utility classes, and the Java compiler output directory.

  The classes in this directory are used by the application class loader to load the classes.

  Folders in this directory will map package and class names.

  The .class files are automatically placed in this directory when the Java compiler compiles the source files from the source directory.

  Any files placed directly in this directory will be deleted by the Java compiler when it runs.

- **webApplication/WEB-INF/lib**: Supports .jar files that your Web application references.

  Any classes contained in these .jar files will be available for your Web application.

In the course of development, the web.xml file will be updated automatically to reflect changes to your Web project. For example, when the New Servlet wizard is used to create a new servlet in a Web project, it will automatically place the appropriate servlet entry into the web.xml file.

4.3.4 Application client project organization

Application client projects allow you to organize your client applications logically.

As you develop client applications in the workbench, your source files will be kept in the appClientModule folder of the application client project and the binary files will be kept in the bin folder.

4.3.5 Server project organization

By default, the following Server Tools resource files are stored in the root of the server project directory. However they may be stored in any directory within the server project.

- WebSphere server instance (.wsi)
- WebSphere remote instance (.wrsi)
- WebSphere remote file transfer (.rft)
- Tomcat server instance (.tsi)
- TCP/IP Monitor server instance (.msi)
- TCP/IP Monitor server configuration (.msc)

A Server project has the following directory structure:

- The templates folder contains the source for server instance templates and server configuration templates. Templates contain settings that may be used as a starting point when creating new server instances or new server configurations.
- WebSphere server configurations are created in folders with a .wsc extension. Each folder has a file called `server-cfg.xml` which contains the setting for that particular server configuration.
- Tomcat server configurations are created in folders with a .tsc extension. Each folder contains the following files that are specific for that particular Tomcat server configuration:
  - The server.xml file contains the base Tomcat server configuration settings.
  - The web.xml file contains the default settings for the Web application.
  - The tomcat-users.xml file contains Tomcat user profile information, such as the Tomcat user IDs with their passwords and a definition of the role of the user ID.
  - The tomcat.policy file contains Tomcat security settings.

5

Programming assists

This chapter describes the main Java programming assist features of Application Developer. These features are designed to make life easier for both experienced and novice Java programmers by simplifying or automating many common tasks.

This following topics are covered:

- Pluggable JDK
- Java snippets
- Code assist
- Import generation
- Tasks view
- Refactoring
- Smart compilation
- Java search
- Bookmarks
- Integrated debugging

© Copyright IBM Corp. 2002

5.1 Pluggable JDK

To provide support for different JDK levels and run-time environments, new JREs can be added to the workbench. For each project you can then select which particular JRE to use. By default the current version of Application Developer supports JDK 1.3 and the corresponding JRE will be used for all projects unless otherwise specified.

See “Choice of JRE” on page 29 for details on how to add a new JRE and how to set the default JRE used for projects.

5.2 Java snippets

Snippets of Java code can be entered in a *Scrapbook* window and evaluated by simply selecting the text and running it. This feature can be used to quickly test code without having to modify any actual Java source files.

Scrapbook pages can be added to any project. They are given an extension of *jpage* to differentiate them from normal Java source files.

To create a Scrapbook page you select: **File—>New—>Other** and select **Java** and **Java Scrapbook Page.** This will bring up the Create Scrapbook page dialog (Figure 5-1).

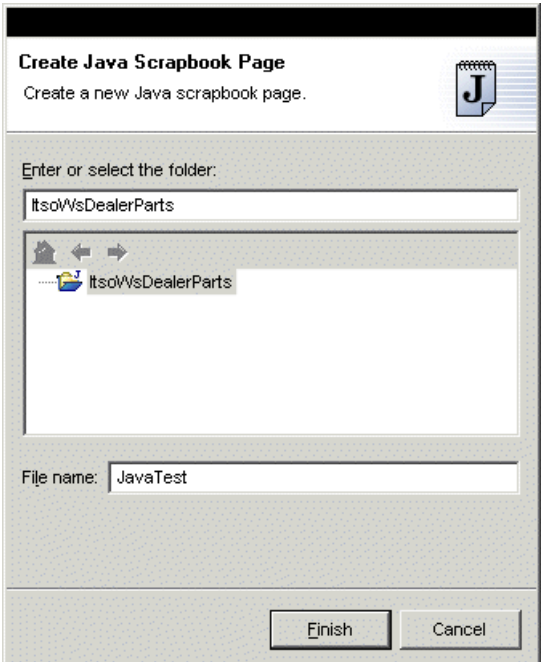


Figure 5-1 Create Java Scrapbook page

You select the folder in which to place the Scrapbook page and enter a name for it.

When the Scrapbook page has been created, you can start entering code snippets into it. Just type the code in and select **Run** from the context menu. Any output from the execution will be shown in the Console window. You can also select **Display** from the context menu to show the return value of the code, or **Inspect** to bring up an Inspector window, which allows you to analyze the contents of the variables in the code.

5.3 Code Assist feature

When writing Java code, you can use the Code Assist feature in Application Developer to display methods and fields that are valid to use in the current context. In the following example, Figure 5-2, you want to use the method from the SQL Connection class to create a new SQL statement, but you can't remember the exact name and parameters. To see the valid methods, position

the cursor at the point where you want to insert the method call and press Ctrl-Space. A window will be shown listing all the available methods and fields. To insert a call to the method createStatement, simply double-click on the method name.

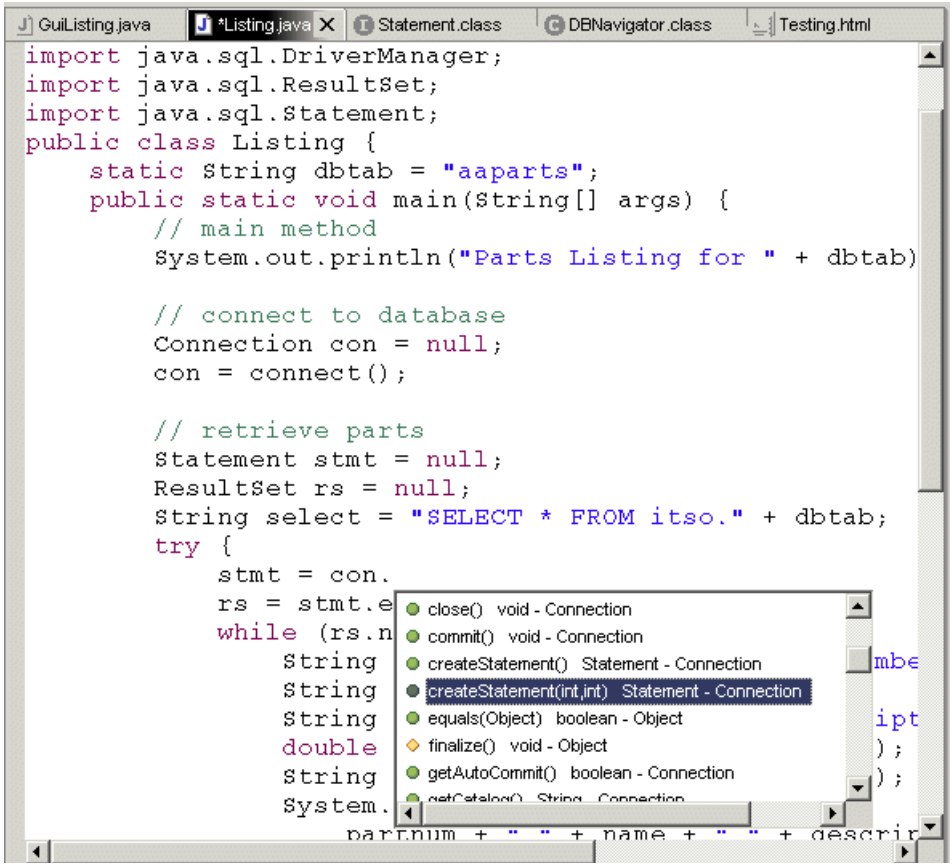


Figure 5-2 Code Assist feature

The Java Editor also supports syntax highlighting and hover-help for variables and methods. The hover-help will show the Javadoc associated with the selected item in the code.

Alongside the Java editor view of your code, there is an Outline view. By selecting elements in the Outline view you can jump to the corresponding point in your code. This allows you to easily find methods and variable definitions without scrolling (Figure 5-3).

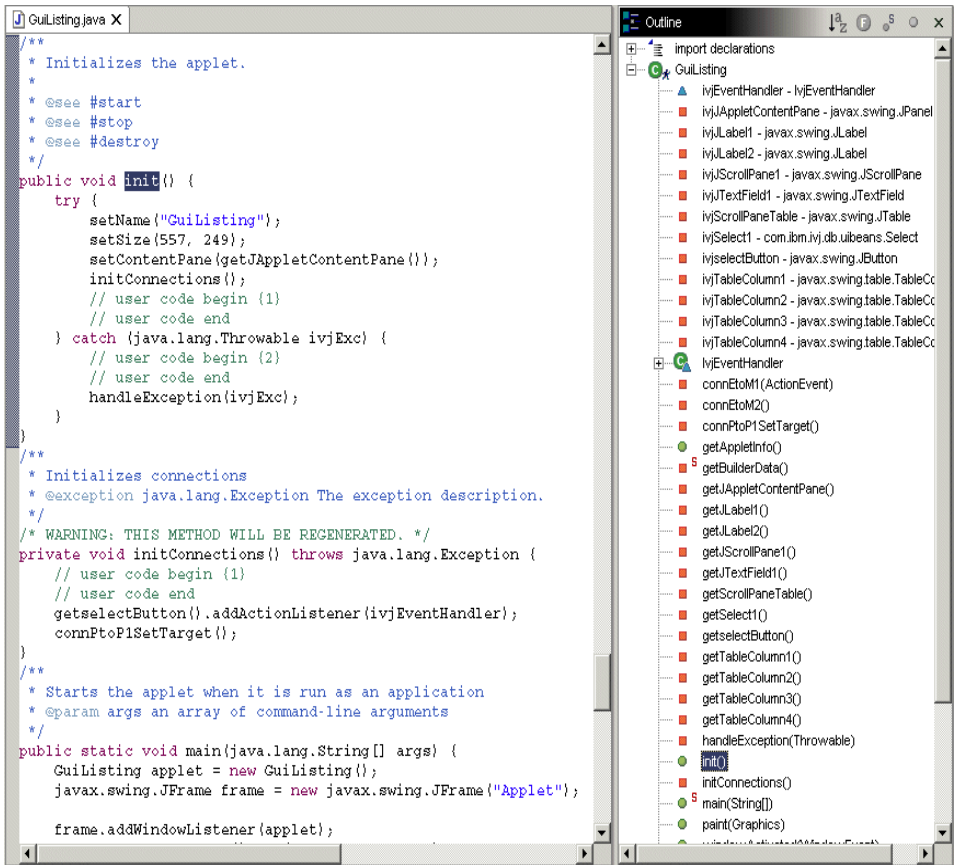


Figure 5-3 Code outline view

Tip: If you have a source file with many methods, you can use the “Show source of selected elements only” icon from the toolbar to limit the edit view to the element currently selected in the Outline view.

5.4 Import generation

The Application Developer Java Editor simplifies the task of finding the correct import statements to use in your Java code. Simply highlight the type name in the code and select **Add Import** from the context menu. If the type name is unambiguous, the import will be pasted in at the correct place in the code. If the type exists in more than one package, a window will be shown from which you

can choose the correct type to generate the import for. Figure 5-4 shows an example where the selected type (Statement) exists in several packages. Once you have determined that the java.sql package is what you want, double-click the entry in the list and the import statement will be generated in the code.

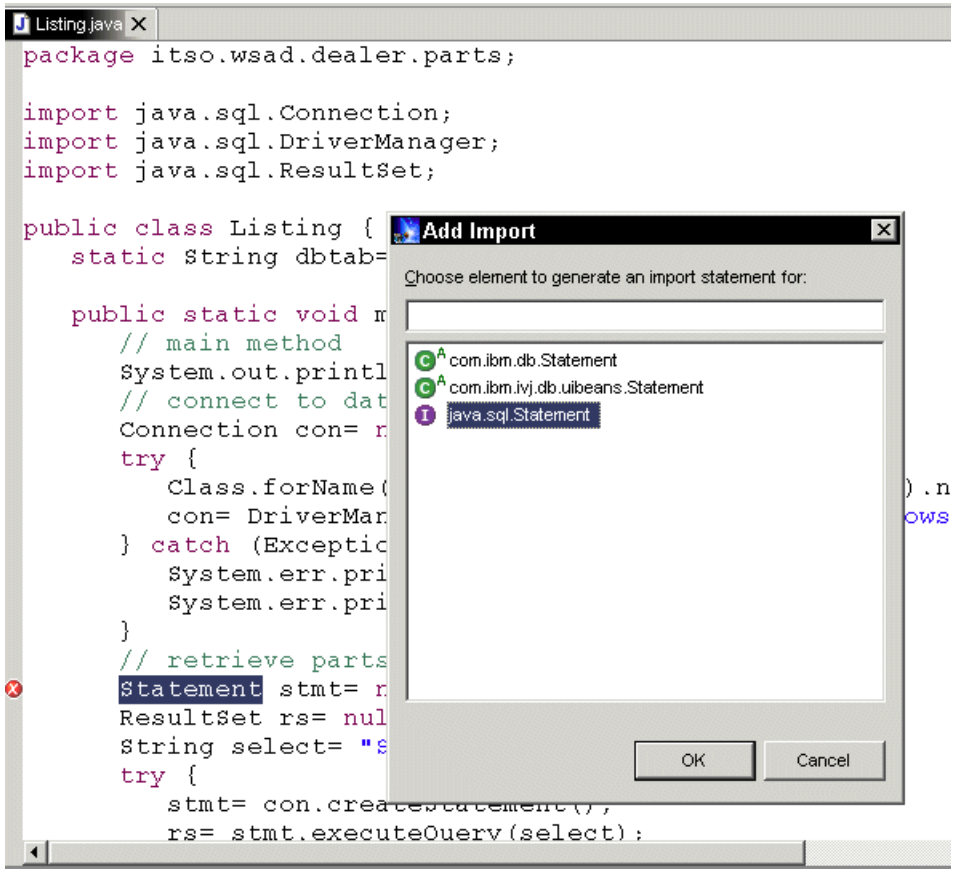


Figure 5-4 Generating import statements

You can also add the required import statements for the whole compilation unit. Right-click anywhere in the Java editor view and select **Organize Imports**. The code in the compilation unit will be analyzed and the correct import statements added. You can control the order in which the imports are added and when package level imports should be used through the Preferences dialog, see “Import organization” on page 27 for details about this feature.

5.5 Tasks view

The Tasks view in Application Developer shows two general types of tasks:

- System generated tasks
- User defined tasks

System generated tasks are typically created by the various builders. As an example any compile errors detected by the Java Builder will be shown in the Tasks view. Such tasks are directly related to a resource, and by double-clicking on the task, the editor on the corresponding resource will be opened.

User defined tasks are global and not related to a particular resource or folder. Tasks in this context are similar to a items in a to-do list. Any user task can be inserted into the Tasks view and tracked for completion.

Figure 5-5 shows an example of a Tasks view with one user defined task and four system generated tasks, (a broken link warning in an HTML file and three compile errors in a Java file).

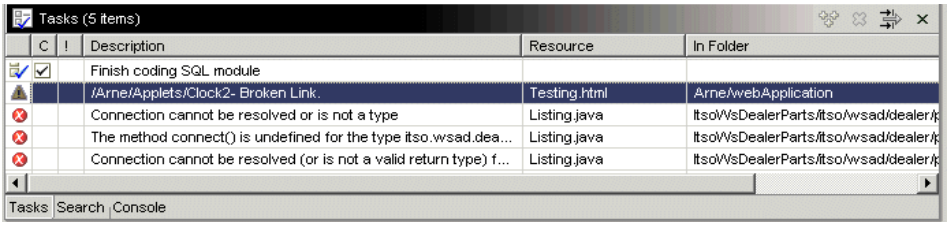


Figure 5-5 Tasks view

The Tasks view can be filtered to show only specific types of tasks, for example you may want to see only errors or tasks related to a specific resource.

5.6 Refactoring

When developing Java code it is often necessary to perform tasks such as renaming classes, moving classes between packages, and breaking out code into separate methods. The term *refactoring* is sometimes used to describe these types of changes. In traditional programming environments such tasks are both time consuming and error prone since it is up to the programmer to find and update each and every reference throughout the project code. Application Developer provides functions to automate this process.

Table 5-1 summarizes the types of refactoring operations supported by Application Developer:

Table 5-1 Supported refactoring operations

Element	Available Refactoring commands
Package	Rename
Compilation unit	Rename. Move to another package
Type	Rename
Method	Rename. Rename parameters
Code	Extract methods

In the **Window—>Preferences—>Java—>Refactoring** dialog you can specify some aspects as to how the refactoring process should work (Figure 5-6).

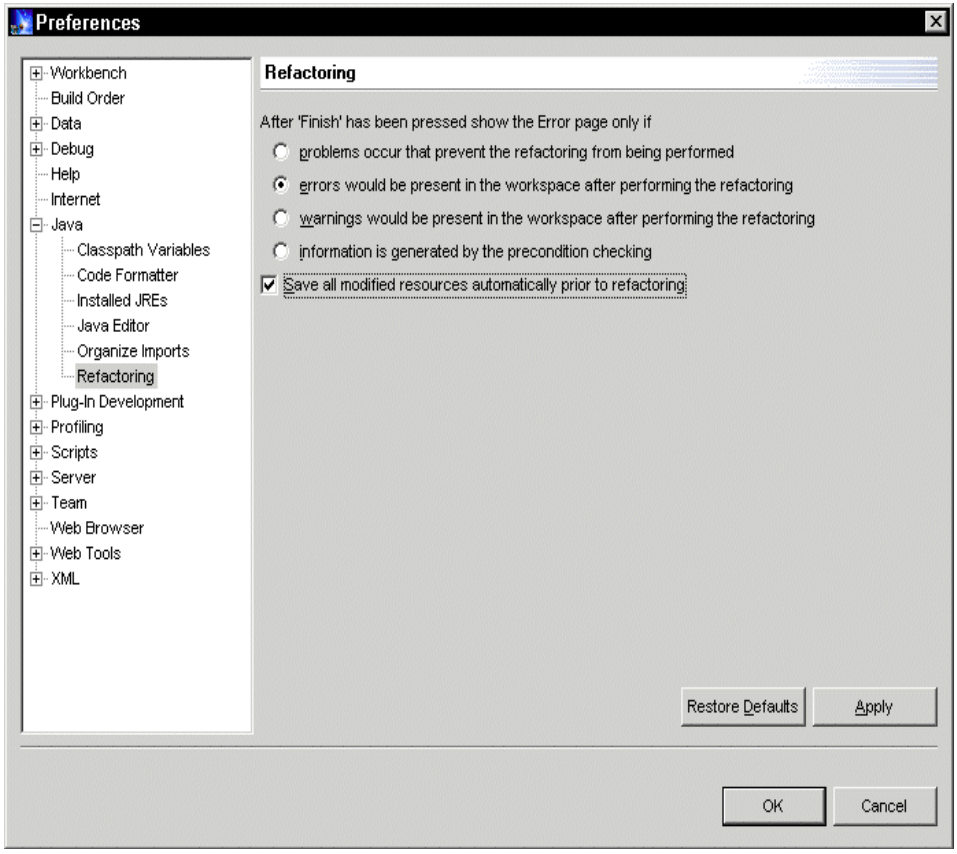


Figure 5-6 Refactoring preferences

The set of radio buttons are used to indicate what type of error reporting you want to see in the Refactoring dialog. The options are listed in order of severity. The default selection is to display any errors that would result if the refactoring was done.

If you check the option **Save all modified resources automatically prior to refactoring,** any outstanding changes will be saved without displaying a prompt.

The following example of a refactoring operation assumes that you want to rename a type in your program. Other refactoring commands work in a similar fashion. To initiate the renaming, right-click on the type in the Outline view. Select **Refactor—>Rename**. The first page of the dialog is displayed (Figure 5-7).

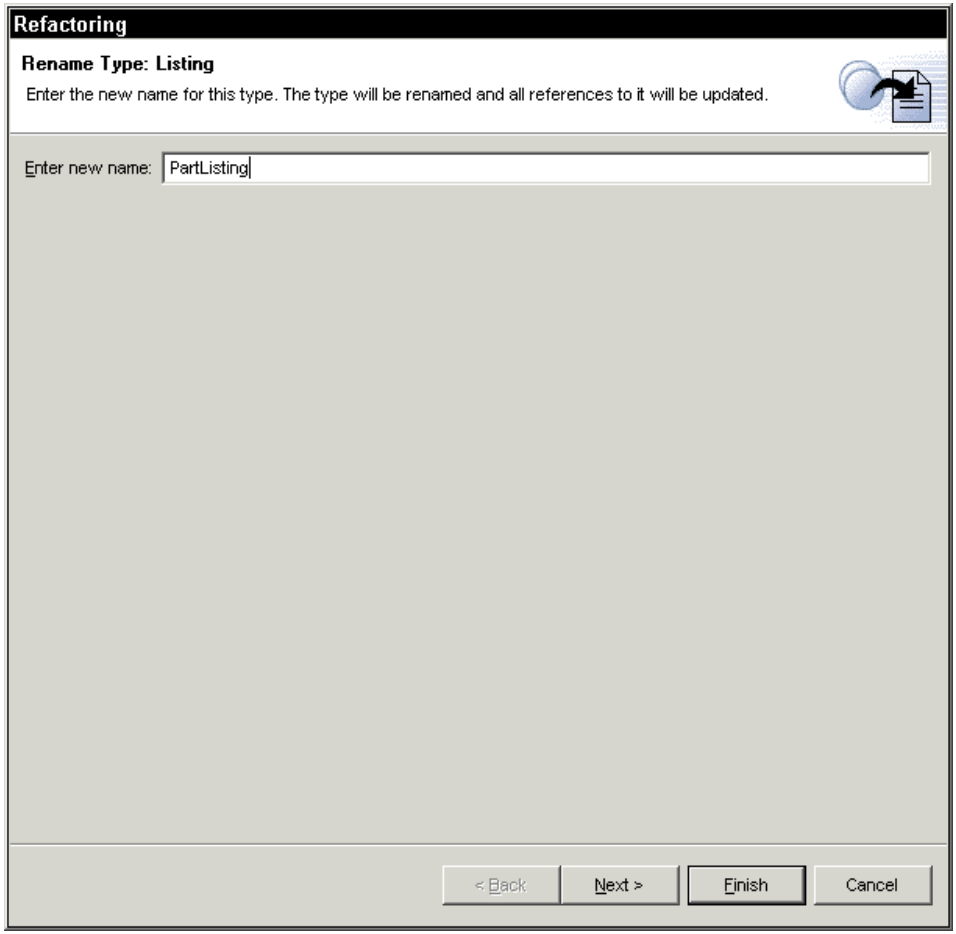

Refactoring
Rename Type: Listing
Enter the new name for this type. The type will be renamed and all references to it will be updated.
Enter new name:
PartListing
< Back
Next >
Finish
Cancel

Figure 5-7 Refactoring page 1

Enter the new name for the Type and click **Next**. If there are any files with unsaved changes in the workbench, and you have not indicated in the Preferences that the save should be automatic, you will be prompted to save these before continuing the refactoring operation.

If problems more severe than the default level set in the refactoring preferences are anticipated, then the problems page comes to the front (Figure 5-8). If the problems are severe, the **Next** and **Finish** buttons will be disabled and the refactoring must be aborted until the problems have been corrected. If the buttons are still enabled, you can select whether to accept the problems and continue, or to cancel the operation.

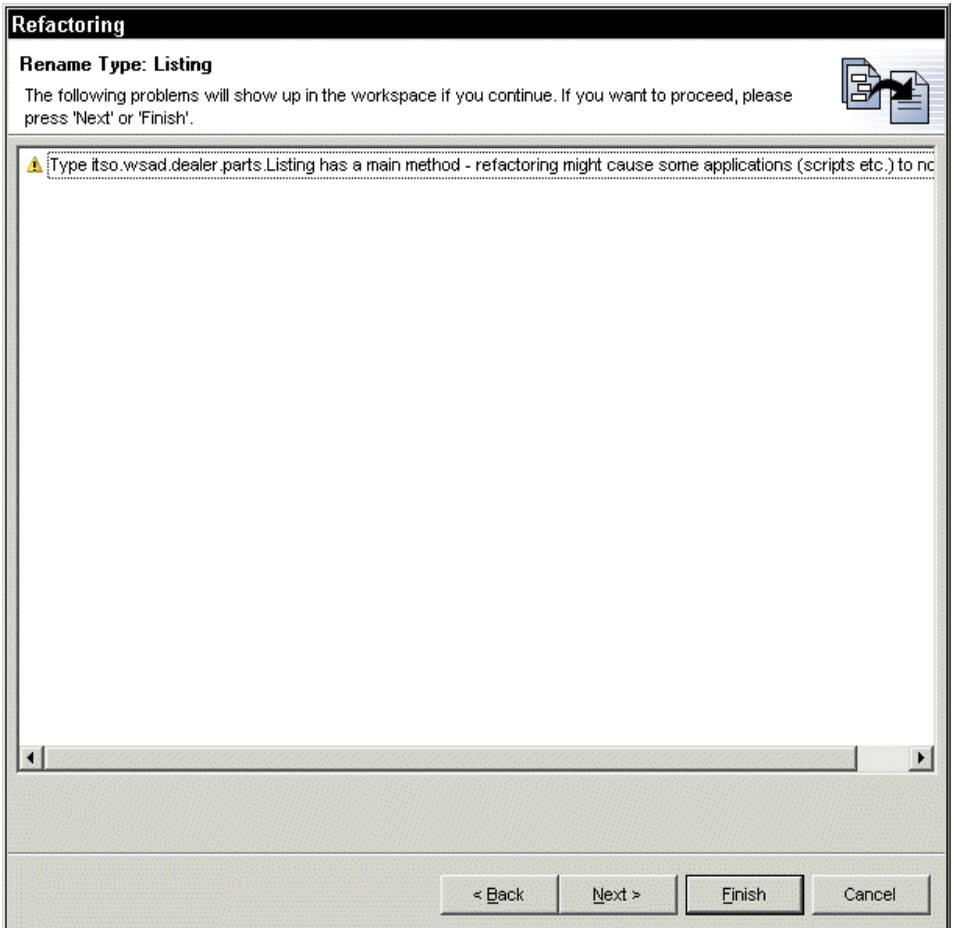

Refactoring
Rename Type: Listing
The following problems will show up in the workspace if you continue. If you want to proceed, please press 'Next' or 'Finish'.
Type itso.wsad.dealer.parts.Listing has a main method - refactoring might cause some applications (scripts etc.) to nc
< Back
Next >
Finish
Cancel

Figure 5-8 Refactoring problems

Selecting **Next** at this point will bring up a window showing what the result of the renaming will be (Figure 5-9).

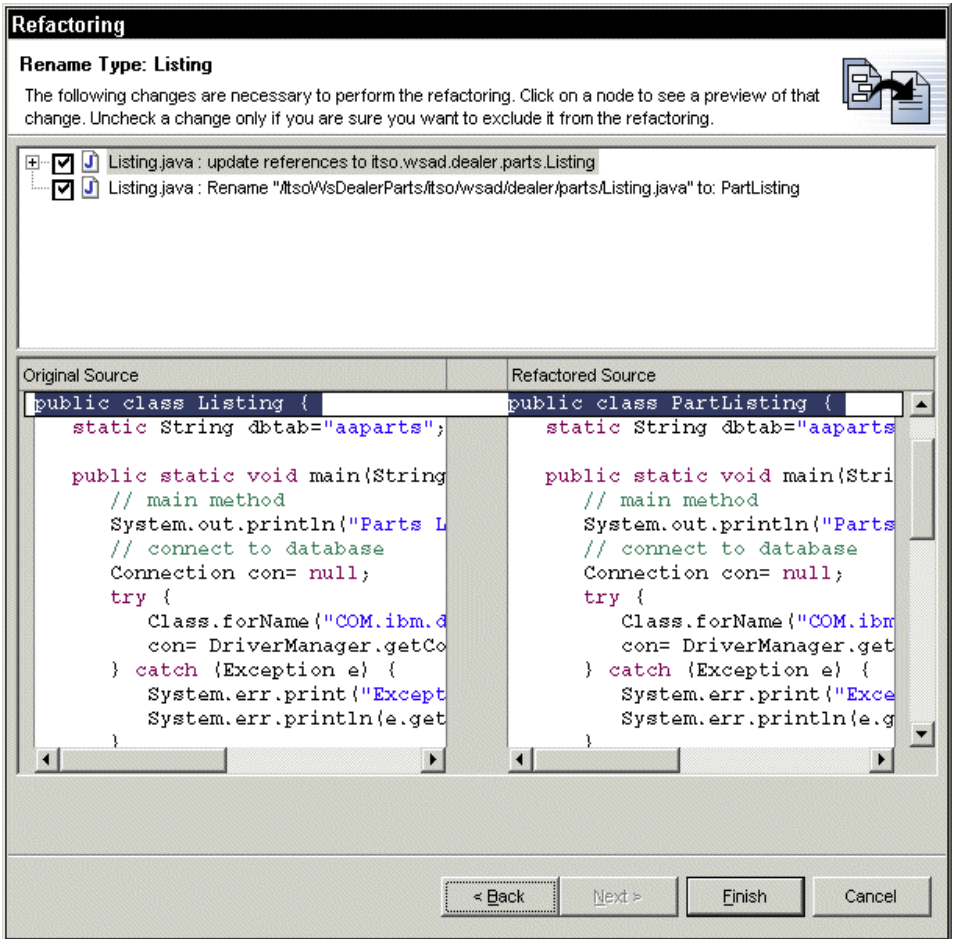


Figure 5-9 Refactoring preview

After reviewing the changes that will be applied, you can again select whether to continue the refactoring operation. Clicking **Finish** at this point will perform the renaming. If there are any problems detected, these will be displayed after the operation has completed. The type of problems shown will depend on your settings in the Preference dialog.

Application Developer provides one level of undo for refactoring commands. If you want to back out the renaming changes at this stage, right-click the type you just renamed in the Outline view, then select **Refactor—>Undo**.

5.7 Smart compilation

The Java Builder in the workbench incrementally compiles the Java code as it is changed. There is no need to ever explicitly invoke the Java compiler.

Note: Because of the extra functionality included in the Application Developer Java Builder, it is not possible to replace it with another Java compiler like *javac.*

5.8 Java search

In addition to the normal text search functionality, Application Developer provides a special Java element search feature Figure 5-10. Using this you can search for types, methods, packages, constructors and fields in the workspace. The search results can be limited to show only declarations, references or implementers. You can invoke the search by clicking on the Search icon in the toolbar, (the "flashlight" icon), or by pressing the shortcut key `Ctrl-H`.

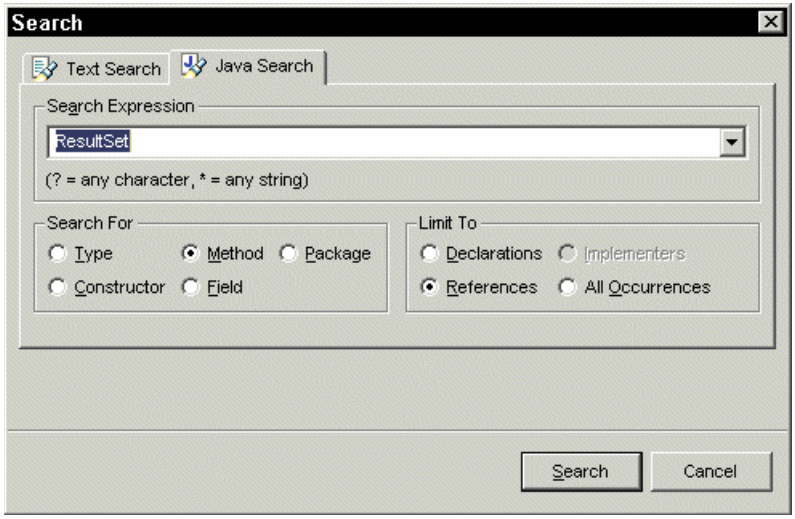


Figure 5-10 Java search dialog

The search results are displayed in the Search view Figure 5-11. To jump to the code, double-click on the line in the results, and the code will be opened in the editor. Yellow arrows are displayed to indicate the lines where a match was found.

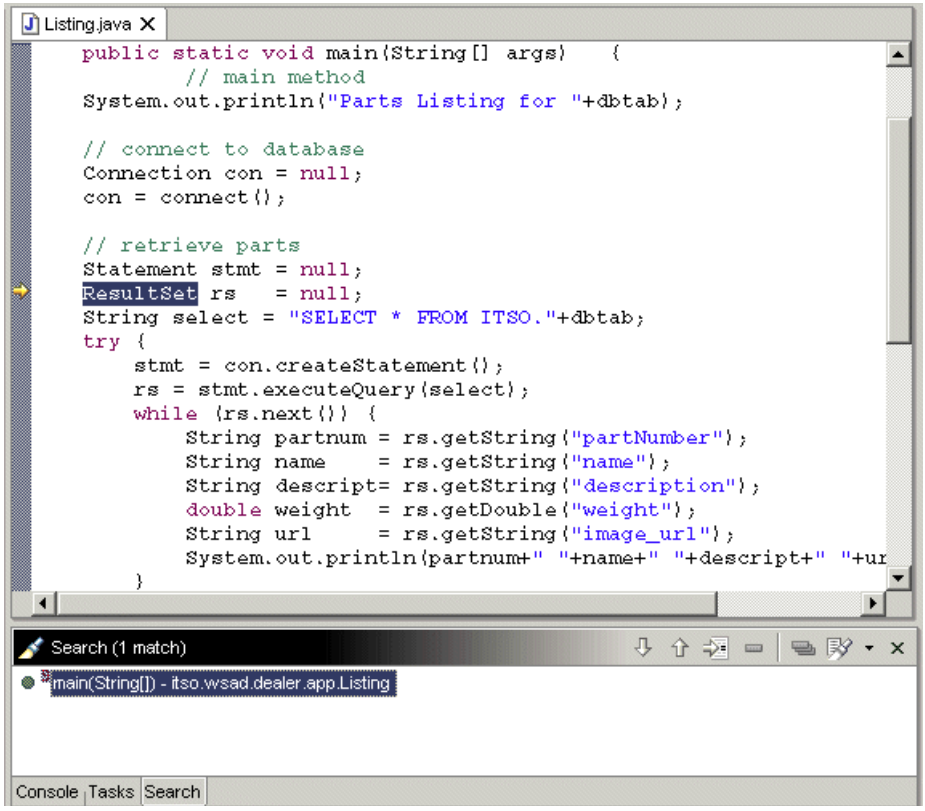


Figure 5-11 Java search results

5.9 Bookmarks

To make it easier to find various parts of your Java code you can create bookmarks. To set a bookmark in your code right-click in the gray sidebar on the left of your code and select **Add bookmark** (Figure 5-12).

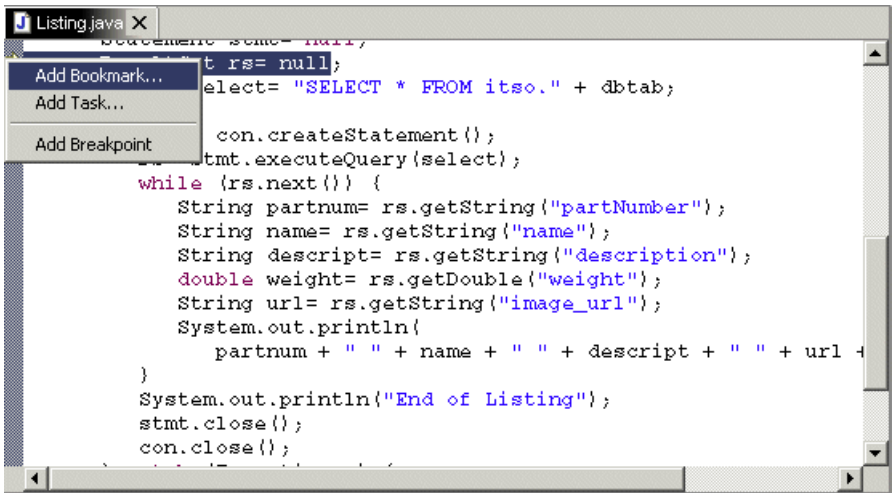


Figure 5-12 Adding a bookmark

A dialog will be displayed where you give a name to the bookmark (Figure 5-13).

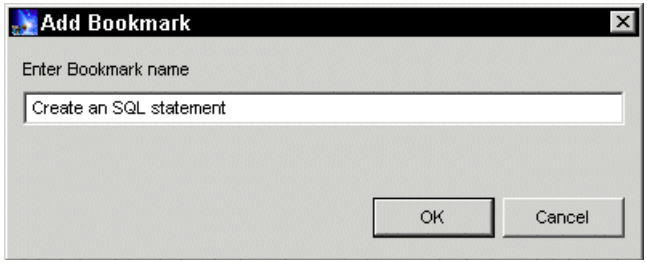


Figure 5-13 Bookmark name

The new bookmark will be added to the Bookmarks view. When you want to return to a bookmark, select **Perspective—>Show View—>Other** and double-click the **Basic->Bookmarks** node. In the Bookmarks view you will see all the bookmarks that are defined, and you can jump to one by double-clicking on it.

Note: Bookmarks are not specific to Java code, they can be used anywhere in Application Developer to provide a quick way of jumping to a specific place.

5.10 Integrated debugging

The Java workbench includes an integrated debugger that makes it easy to trace the execution of Java code and to quickly find and fix problems. You can debug standalone Java code, servlets, EJBs and JSPs and debugging can be done locally or remotely. Debugging is described in detail in “Debugging your code” on page 157 and is also discussed in “Testing and debugging your application” on page 393.

Part 2

Developing applications

Part 2 describes, step-by-step, the various patterns of application development.

© Copyright IBM Corp. 2002

6

Creating Java applications

This chapter discusses the following topics:

- Java development in Application Developer
- Creating a Java project
- Creating Java packages and classes
- Running your code
- Locating compile errors
- Debugging your code
- Java development in Application Developer versus VisualAge for Java

To illustrate these concepts we will step through the development of a simple Java class that reads a DB2 database table to retrieve information about car parts and displays the results back to the user. Here we will show how to do this as a standalone Java application. In later chapters we will discuss how to do the same thing using a Java Applet and a JavaBean. The Java application uses JDBC to interface to the database, but the details of this are not discussed in this chapter. Database issues are discussed extensively in Part 3 of this book.

Note: To be able to run the sample Java application you first need to create and populate the DB2 tables as described in Appendix C, “Additional material” on page 675.

© Copyright IBM Corp. 2002

6.1 Java applications

You can use Application Developer to develop the Java packages for your application specific business logic and for common code that will be used by several projects. These packages can be exported as JAR files and included in the build path for other types of projects, or in the application class path of the application servers.

Existing Java code can also be imported and integrated into new applications. If you are currently using VisualAge for Java, you can export code from there and import it into Application Developer for integration with the rest of the project components.

Application Developer provides tool support for the following development tasks:

- Creating a Java project
- Editing code
- Refactoring code
- Searching code
- Building code
- Running code
- Debugging code

6.2 Creating a Java project

To create a new Java project select **File—>New—>Project**. This will display the New Project dialog (Figure 6-1).

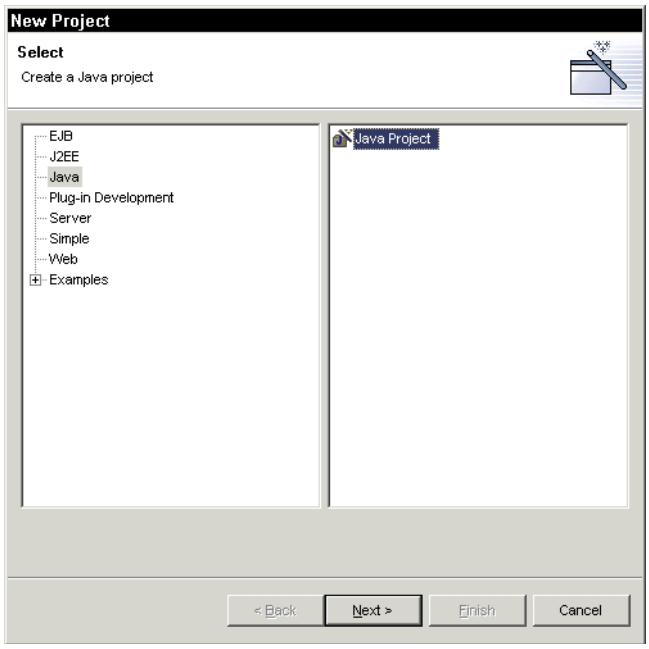


Figure 6-1 Starting the Java project wizard

Select **Java** and **Java Project** from this dialog and click **Next** to start the Java Project Wizard (Figure 6-2).

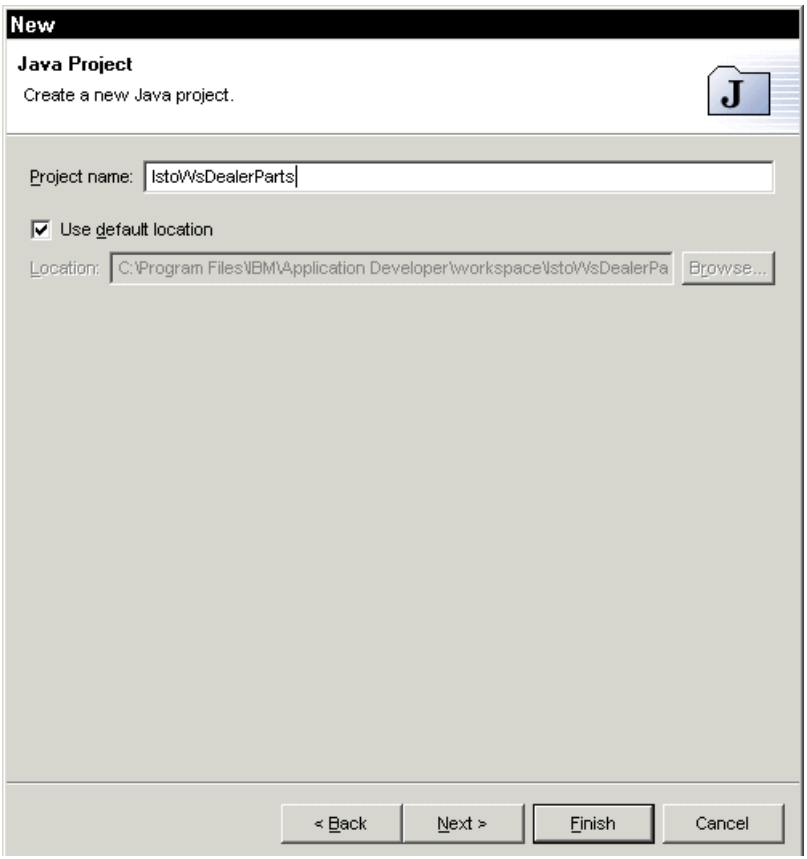


Figure 6-2 Java project wizard - page 1

On the first page of the wizard you give a name to the project, (`ItsoWsDealerParts` in this example), and the directory on the file system where the project files should be stored. By default they will be stored in a directory created under the Application Developer workspace directory. Clicking **Next** brings up the second page of the wizard where you define the Java build settings for the new project (Figure 6-3).

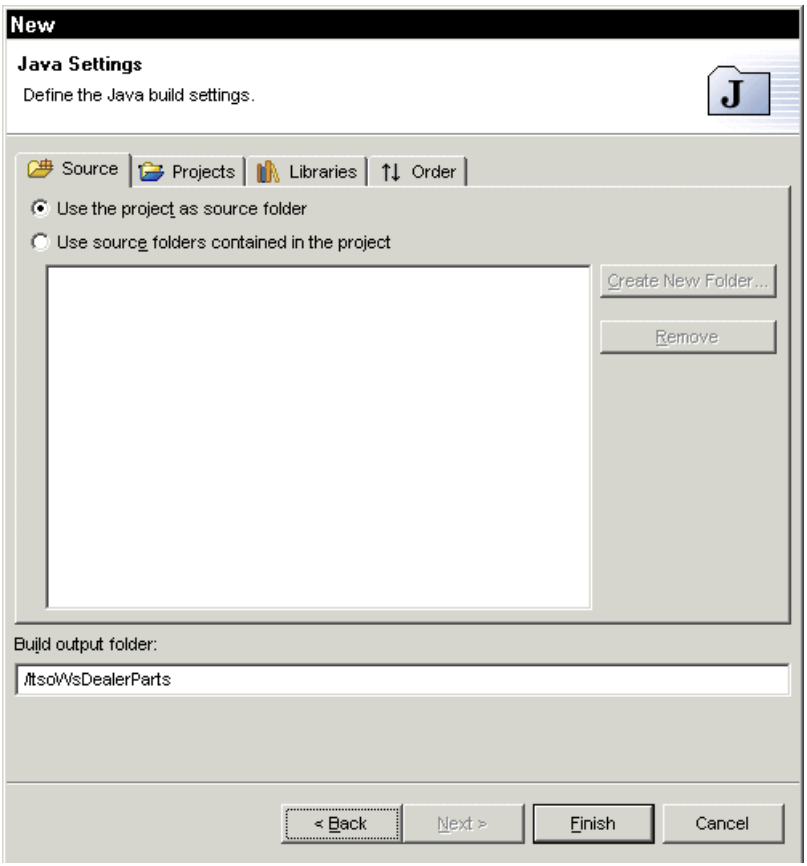


Figure 6-3 Java project wizard - page 2 (Source tab)

On the Source tab you decide whether it is appropriate to store the source code directly in the project folder, or if you want to use separate source folders. For our sample project the simple model is used. If you want to use the complex model instead, you can create the required folders by clicking **Create New Folder** and adding them to the list.

Here you can also select the target folder for the generated class files. By default they will be placed directly in the Project folder, but you can edit the **Build output folder** field to define another target folder.

Note: In the packages view you can't actually see the generated .class files, but if you look in the file system you will see that they are physically stored in the directory you specify when creating the project.

On the Projects tab you can specify any other projects in your workspace that should be in the Java build path for the new project (Figure 6-4). You may have some common code in a project that already exists that you want to reuse in the new project.

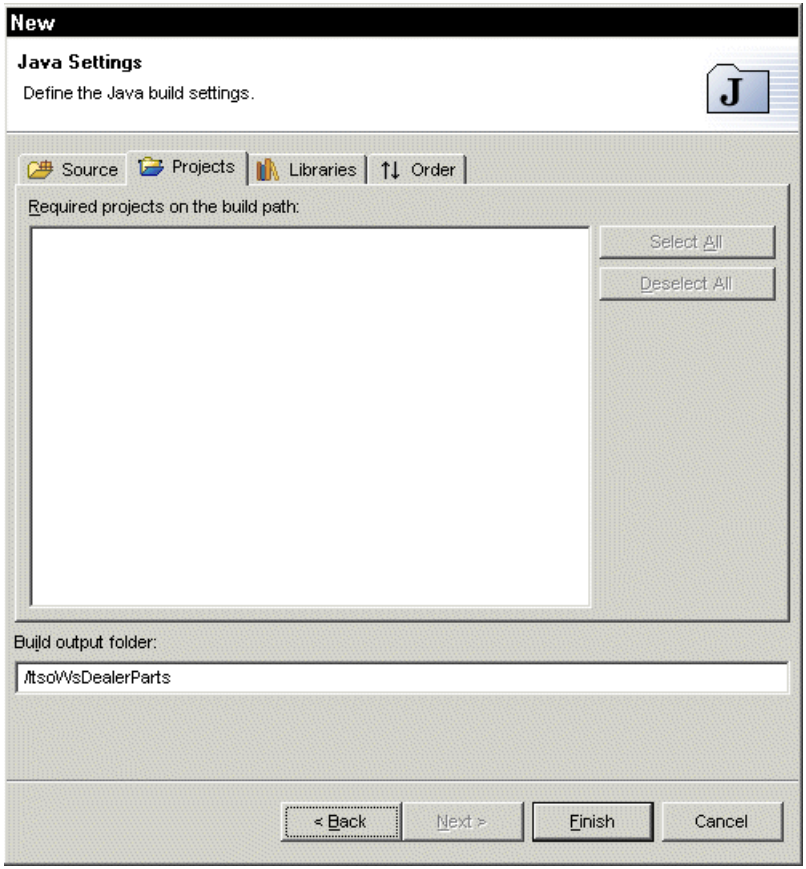


Figure 6-4 Java project wizard - page 2 (Projects tab)

On the Libraries tab, you can add in any other code libraries that need to be included in the build path (Figure 6-5). By default, only the JRE library will be included. If required, you can add internal JAR files, that is, JAR files that have been imported into the workbench, or external JAR files that you reference from the file system. You can also add in Java classpath variables that were defined when your workbench preferences were set up, see “Defining Java class path variables” on page 24 for instructions on how to create such variables.

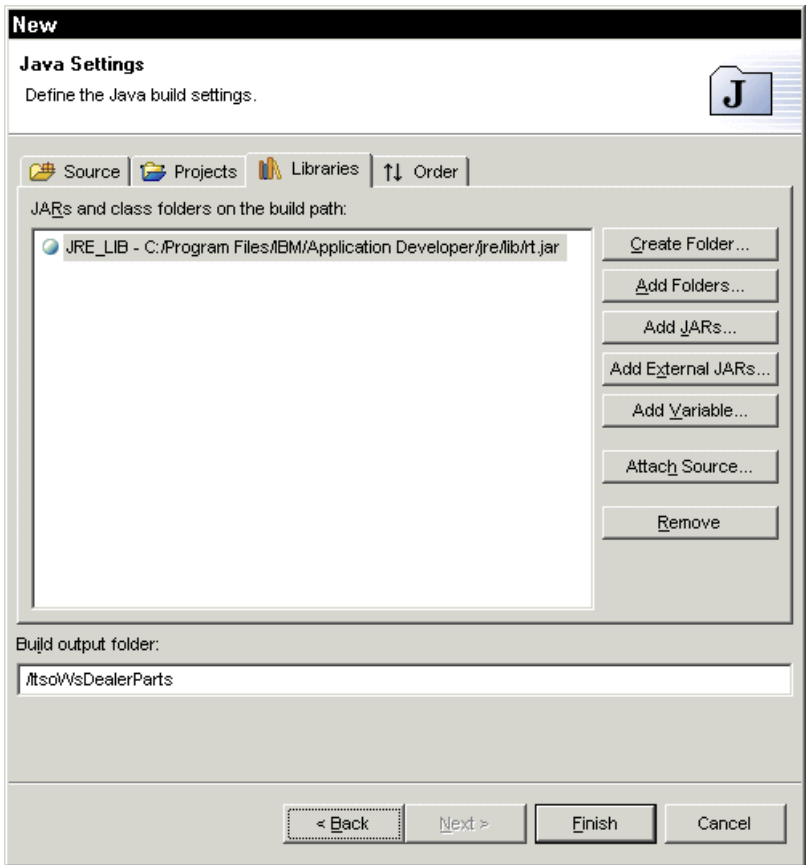


Figure 6-5 Java project wizard - page 2 (Libraries tab)

On the final tab, Order, you can specify the order in which you want items in the build path to be searched. Use the **Up** and **Down** buttons to move entries in the list (Figure 6-6).

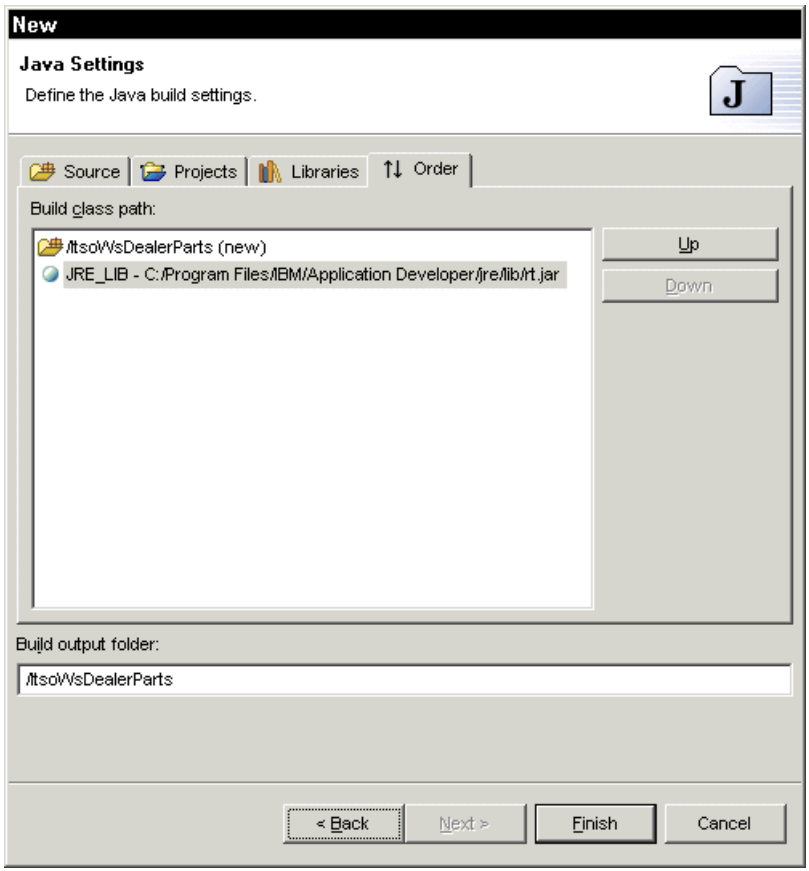


Figure 6-6 Java project wizard - page 2 (Order tab)

Clicking **Finish** will create the new project.

6.3 Create Java packages and classes

Once the project has been created, you can add Java packages to it. This is done by selecting the new project in the Packages view and selecting **New—>Package** from the context menu. In the dialog window, you enter the fully qualified name of the package. The package where the sample Java application code resides is called *itso.wsad.dealer.app* (Figure 6-7). Press the **Finish** button to continue.



Figure 6-7 Create Java package

Once a package has been created you can add classes to it. The sample code is in a class called *Listing*. To create the class, select the package that was created in the previous step and select **New—>Class** from the context menu (Figure 6-8). In the dialog window, you enter a name for the class and which superclass to inherit from.

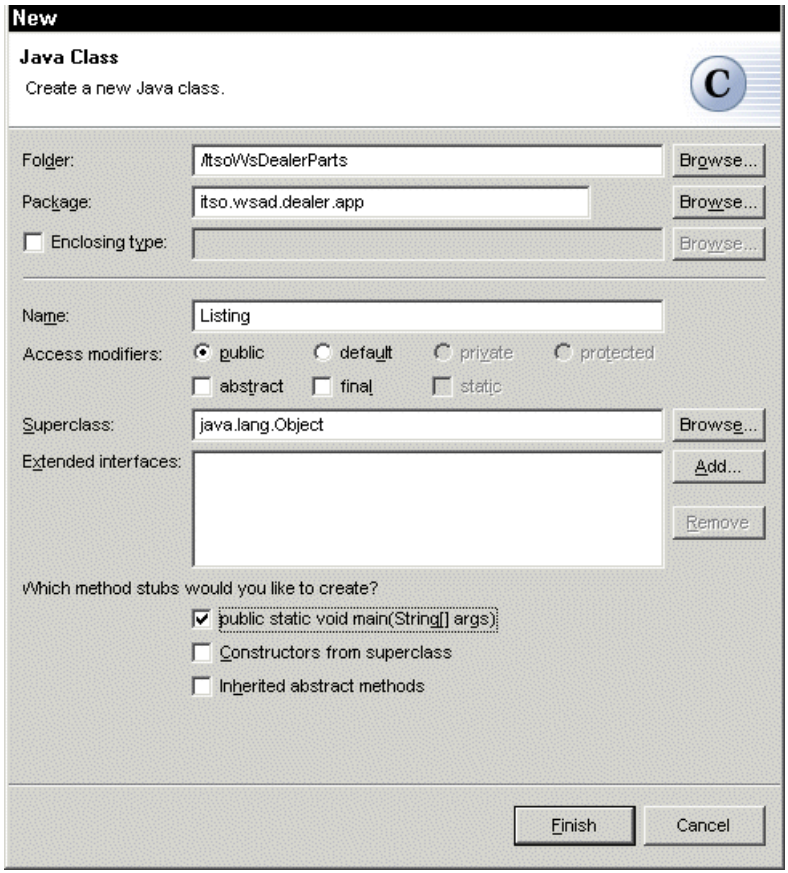


Figure 6-8 Creating Java class

You can also specify the access modifiers for the class, any interfaces that are implemented, and any method stubs that you want created. In our example we choose to create a stub for the main method since a standalone Java application must have one. Once you are done, you click **Finish** to create the class. The Java editor will open.

At this stage you may want to paste in the complete code for Listing.java. If you have installed the sample code for this book according to the instructions in Appendix C, “Additional material” on page 675, you will find Listing.java in:

```
..\Chapter6\itso\wsad\dealer\app
```

The completed code is shown in (Figure 6-9).

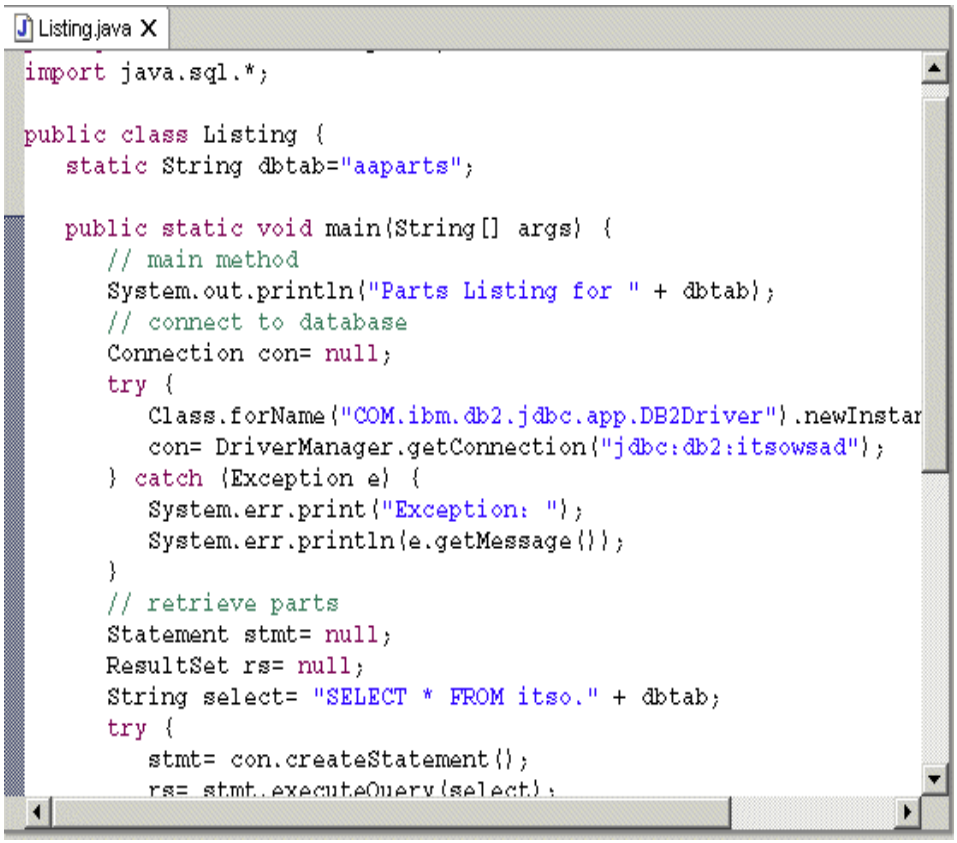


Figure 6-9 Java source editor

We will not explain in detail the code for the Listing class. The basic functions of the sample code are:

- Connect to the DB2 database
- Select all parts from the parts table
- Display each part to the user using the System.out class

6.4 Running your code

Once the code has been completed and is free of compile errors, it can be executed using the workbench Java Application Launcher. To launch the application, you select the **Run** icon  from the toolbar. If this is the first time that you launch a Java class in this project, you will be prompted to select what launcher you want to use (Figure 6-10). Since you want to execute a

standalone Java application, you choose **Java Application** from the list. On subsequent launchings this will be the default for this class. You can also make it the default for the whole project by checking the box **Set as default launcher for project**.

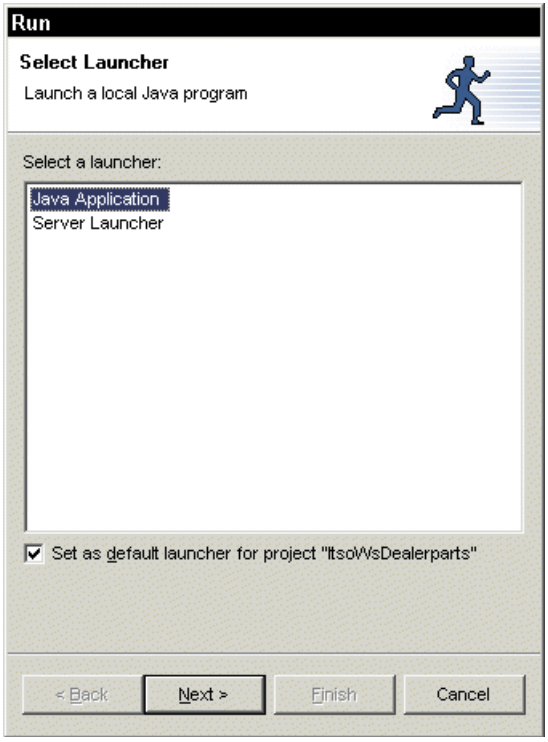


Figure 6-10 Select launcher for Java application

Press **Next** and **Finish** and the main() method of the Listing class will be invoked. The output will be displayed in the Console view.

By default Application Developer will switch to the Debug perspective when you run Java code. In many cases you might prefer to stay in the Java perspective to run and test your code. To change this behavior, select **Window —> Preferences—>Debug** and un-check the **Show Debug Perspective when a program is launched in run mode** check-box (Figure 6-11).

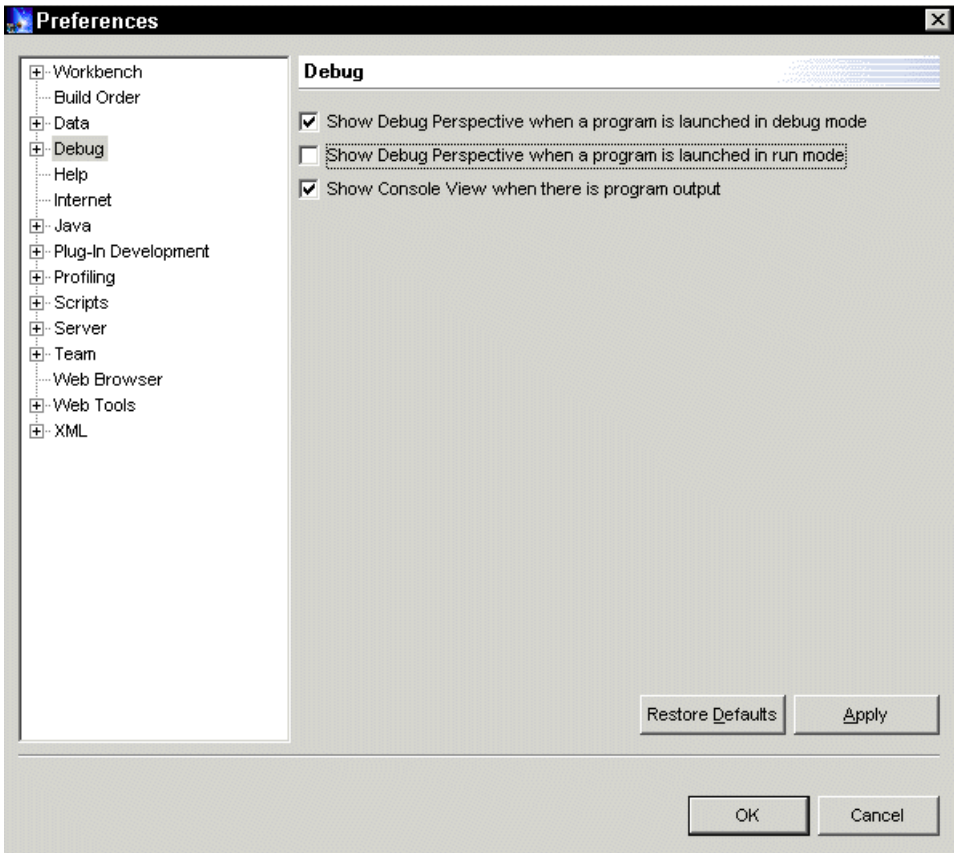


Figure 6-11 Setting preferences to run programs in Java perspective

With the Java Project properties defined as described above, executing the Listing class code should result in an error. The Debug perspective will open up and an error message will be displayed in the Console view:

```
Exception: COM.ibm.db2.jdbc.app.DB2Driver
Exception: null
```

The exception information indicates that there was a problem locating a class required to run the program. To correct the error you need to make sure that you have access to the Java class where the DB2 JDBC Driver code is located. To do this, you have to go back and update the Java build path that was previously defined. You do this by selecting the Java project and selecting **Properties** from the context menu. Then you select the **Java Build Path** node and the **Libraries** tab (Figure 6-12).

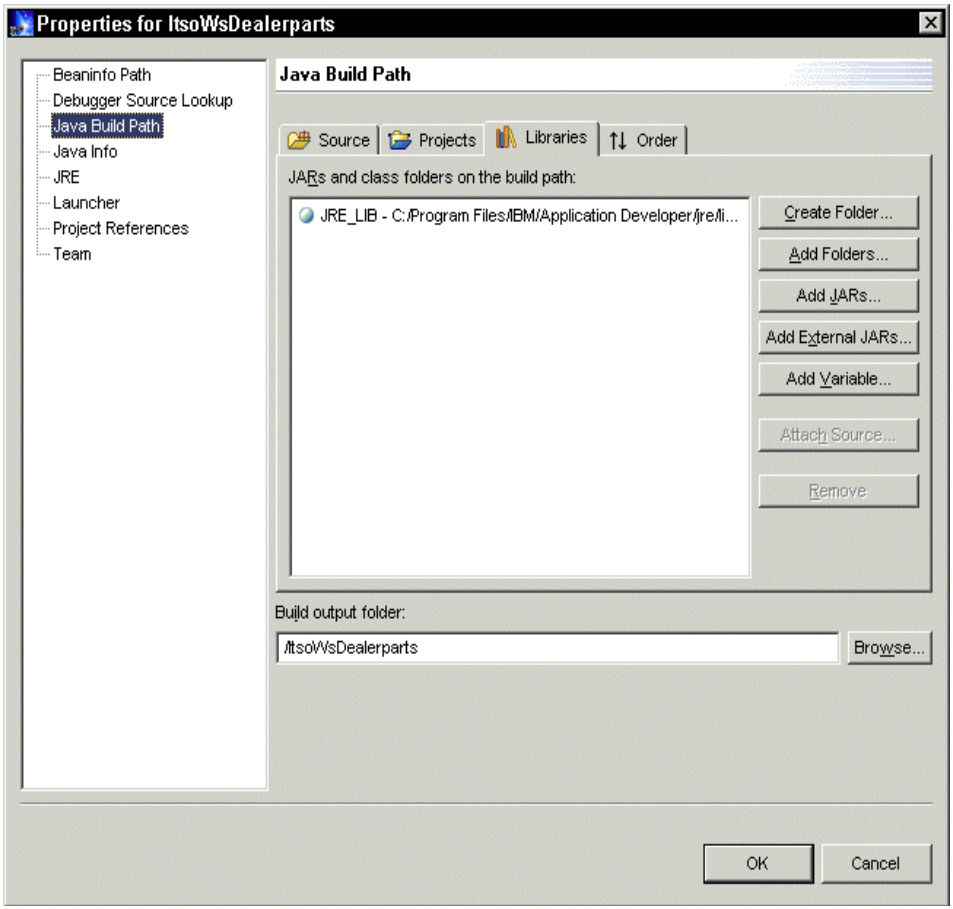


Figure 6-12 Add library to Java build path

There are two ways you can specify access to the required classes:

1. Selecting **Add External JARs** and locating the file, (DB2JAVA.ZIP), on the file system.
2. Selecting **Add variable** and adding an indirect reference to the file.

It is recommended to use the second option since this means that you are not directly referencing any physical path that could be different for another developer. For the sample it is assumed that a variable called DB2JAVA was defined when the workbench setup was done. To add this variable to the Java build path for the project, select **Add Variable** to display the Classpath Variable Selection dialog (Figure 6-13).

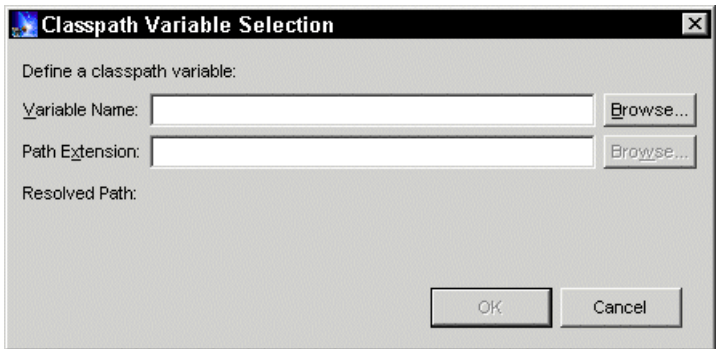


Figure 6-13 Java build path variable selection dialog

If you don't know the name of the variable, you can click **Browse** and a list of all defined variables will be displayed (Figure 6-14).

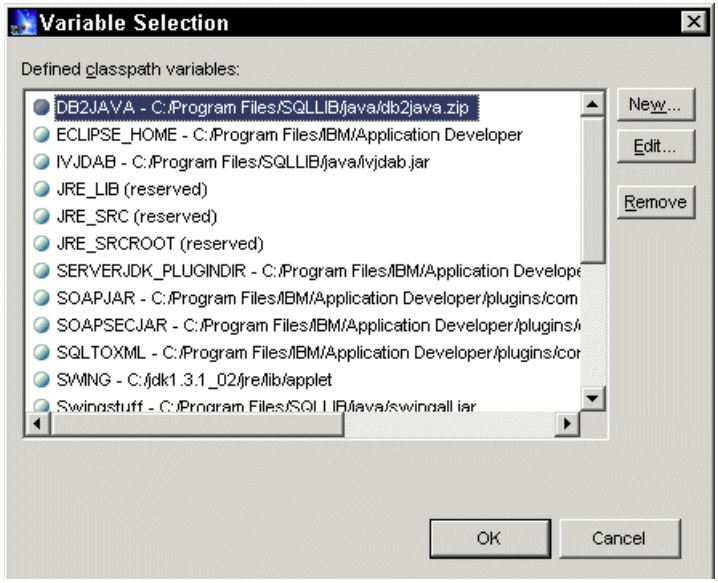


Figure 6-14 Class path variable selection dialog

When you have found the one you want, you can add it to the build path by clicking **OK** and then **OK** again in the previous dialog.

The Listing code can now be run again and the list of parts in the database table should be displayed in the Console view.

If you would like to run this application outside of Application Developer you can export the class file to the file system and run it from there. To do this you need to follow these steps:

Select **File—>Export**. The Export dialog will be displayed Figure 6-15.

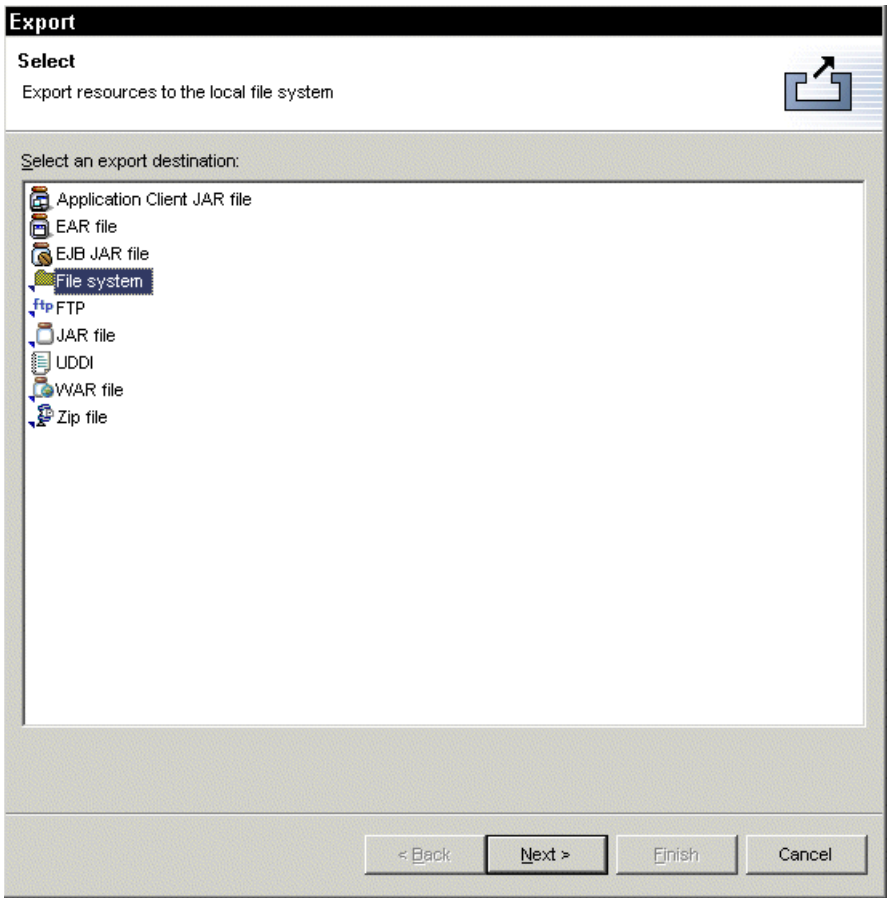


Figure 6-15 Export dialog

This dialog allows you to select the destination of the export. For the purposes of this sample you can do a simple export to the file system. If the code had been in several class files, or if there had been other resources required to run the program, you could have chosen to export as a JAR file instead. When you have made your selection press **Next** to specify resources to export (Figure 6-16).

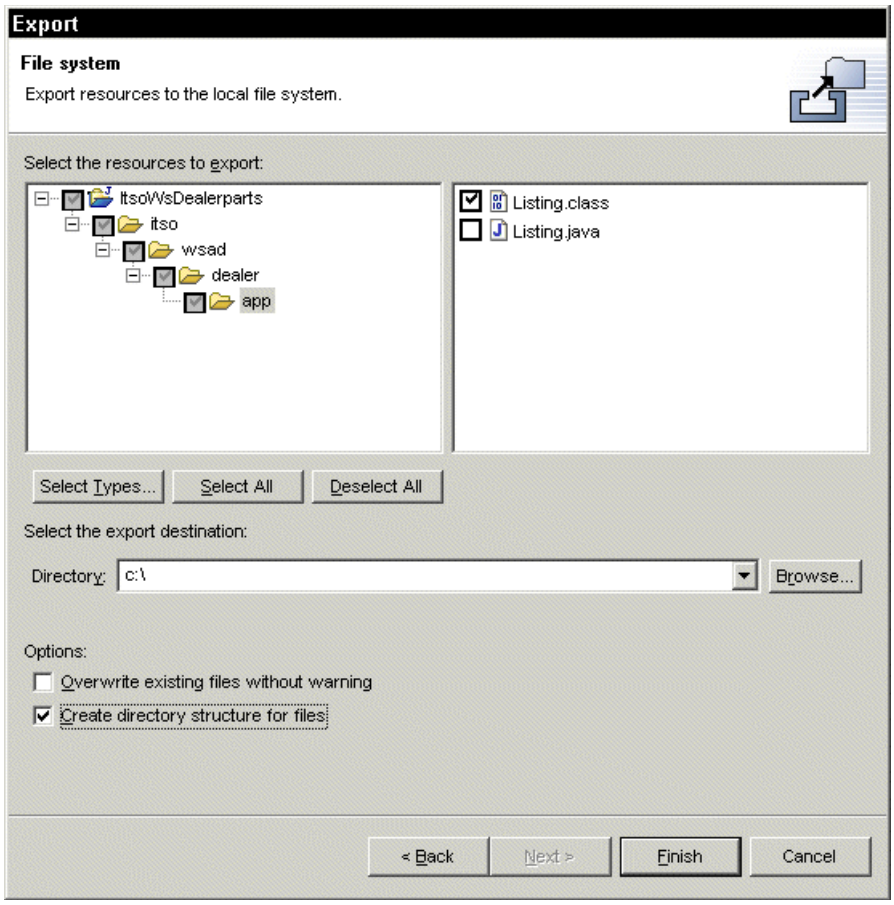


Figure 6-16 Select files to export

Expand the project and select the Listing.class file. Then enter the directory path where you want the file to be stored. In the example we specify `c:\` as the path and check the **Create directory structure for files** check box. This will create the directory path `.\itso\wsad\dealer\app` and place the file there.

To run the application, go to a command window and enter the following command:

```
java -cp ".;c:\program files\sqllib\java\db2java.zip" itso.wsad.dealer.app.Listing
```

Note: The command may be different on your system depending on where your DB2 UDB installation is.

You also need to ensure that java.exe is accessible in the path. In the Application Developer installation you will find a copy of java.exe in ...\Application Developer\jre\bin

3. The listing of parts should be displayed.

6.5 Locating compile errors in your code

All compile errors in your Java code are shown in the Task view. In Figure 6-17 you can see an example of an error in the Listing.java file. The line where the error was found is indicated by a red "X" in the left margin. Double-clicking on the entry in the task list will take you to the line where the error was detected.

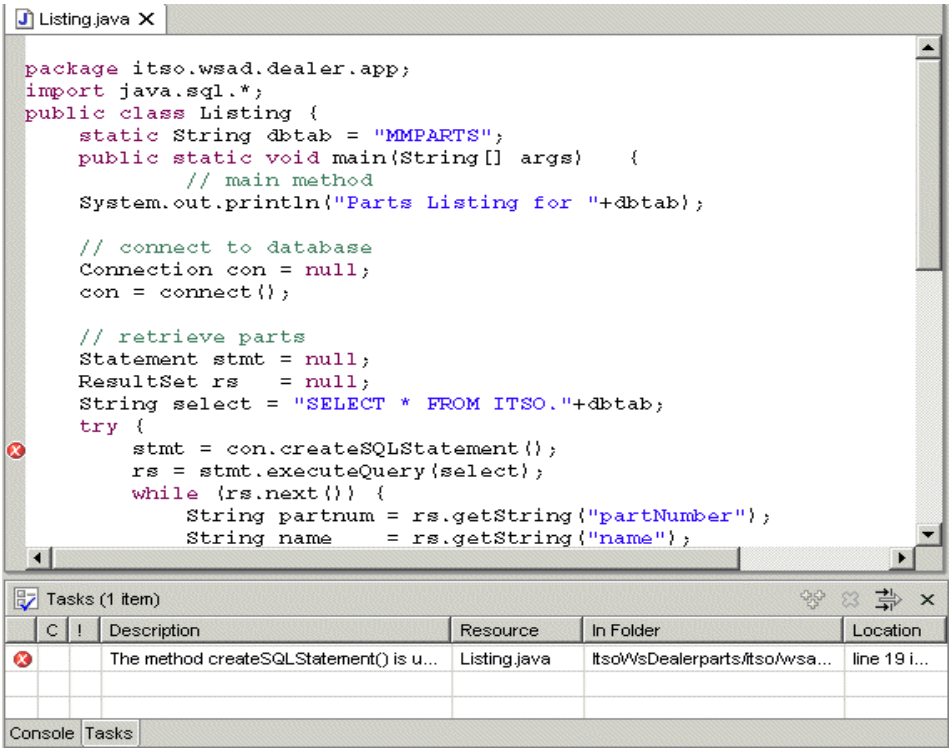


Figure 6-17 Identifying errors in Java code

To more easily find the errors in the file you're working in, you can filter the Task view to only show errors related to the currently selected resource. To do this click the **Filter** icon  in the Task view and select the **On selected resource only** option in the Filter dialog.

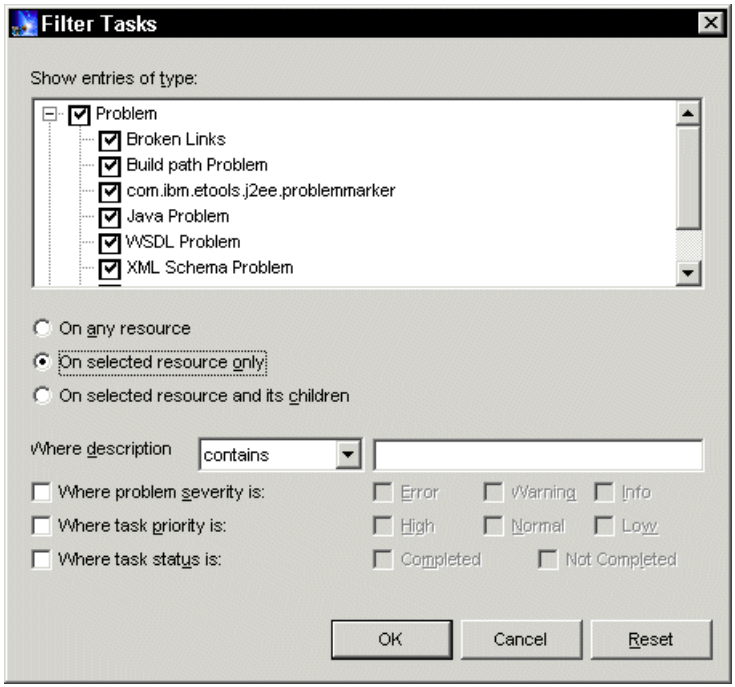


Figure 6-18 Show errors on selected resource only

6.6 Debugging your code

How to debug Java applications in Application Developer is described in detail in Chapter 17, "Testing and debugging your application" on page 393. Here we will quickly look at how you would go about debugging the simple Java application we have just created.

To set a breakpoint in the code, display the code listing and click in the grey area at the left of the statement where you want the execution to stop. Figure 6-19 shows the Listing.java code with a breakpoint being set at the start of the while loop. If you want to see all breakpoints that are currently set, select **Perspective—>Show View—>Other** and select the **Breakpoints** view under **Debug**.

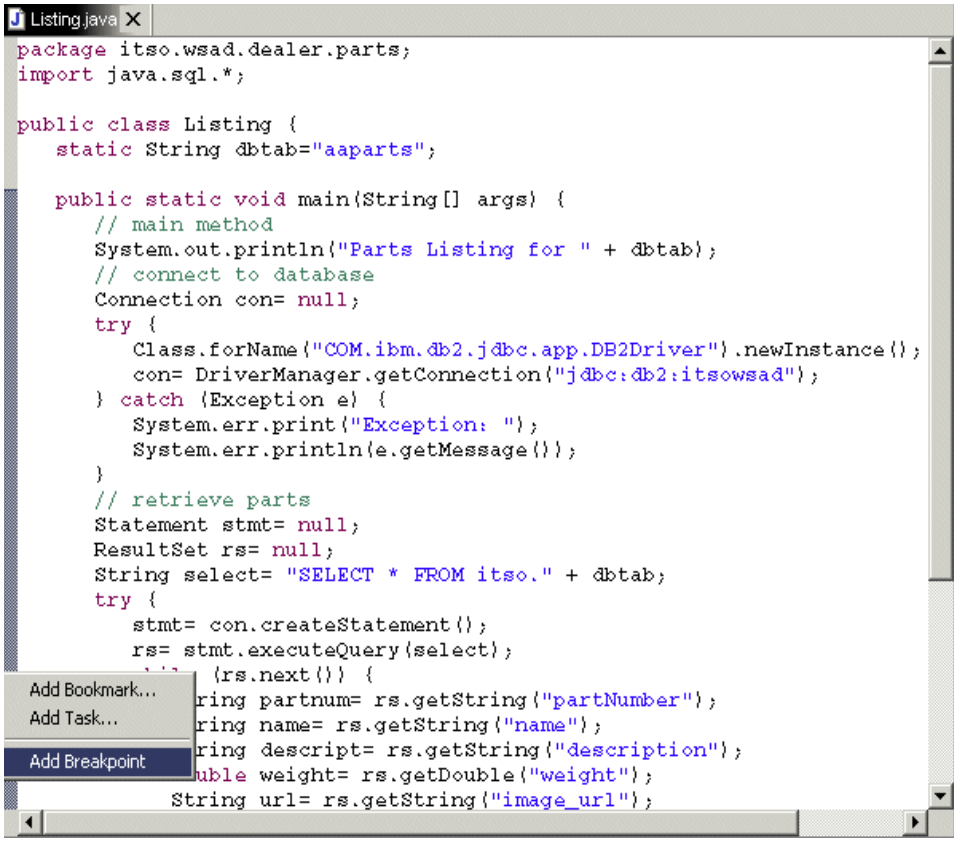


Figure 6-19 Setting a breakpoint

When you have set the breakpoints, you can start the program in debug mode by clicking the debug icon  in the toolbar.

The program will run until the first breakpoint is reached and then display the Debug perspective. From this you can view, inspect and modify variables and single-step through your code (Figure 6-20).

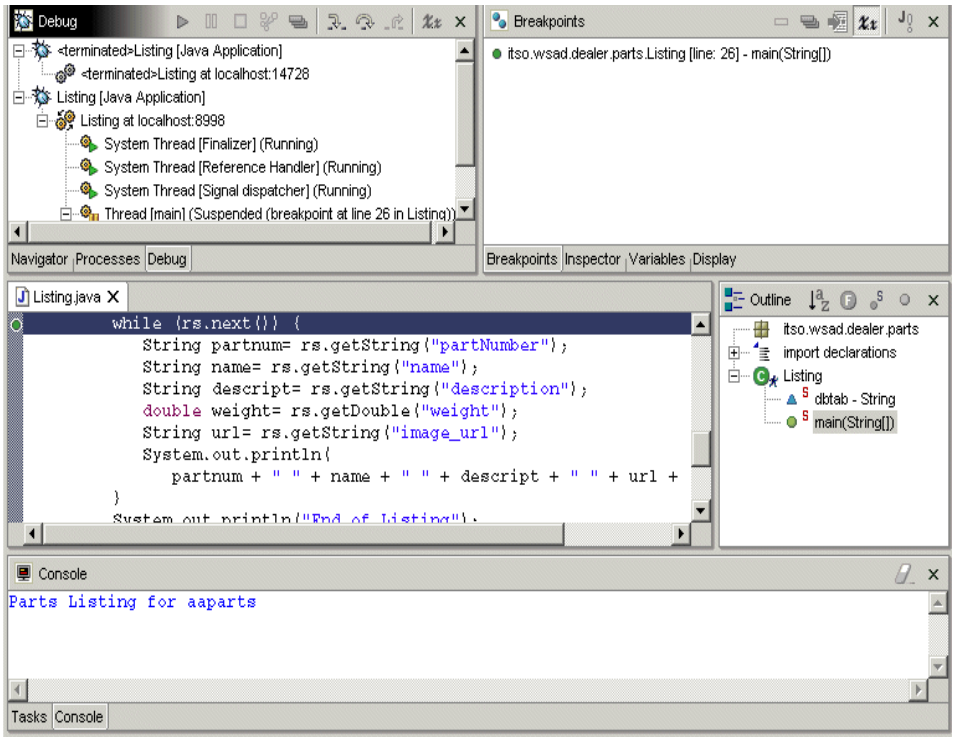


Figure 6-20 Debug perspective

6.7 Java development in Application Developer

Application Developer is the strategic replacement for VisualAge for Java. Even though there are many similarities between developing Java code in the two environments, there are some differences you should be aware of.

- **Storing source code**

  In VisualAge all the source code is stored in a propitiatory repository called Envy. Envy provides Source Configuration Management, (SCM), and keeps track of all code changes down to the method-level.

  In Application Developer source code is stored in `.java` files in the file system. SCM is handled by separate tools such as CVS or ClearCase. *These tools only work at a file level granularity, thus method editions are not supported*.

- **Class bytecodes**

  In VisualAge the classes used for the compilation context is the set of loaded classes inside the workspace. The JDK inside the workspace can't be changed.

  In Application Developer the classes used for compilation are specified in the project buildpath. The user can control this path and can include external JAR files and file system paths. The JRE that is to be used is also a project-setting. Any JDK level from 1.1.7 to 1.4, along with various custom VMs, is supported.

- **Build paths**

  In VisualAge each class is configured with a project path, that is you have to tell the compiler from which other projects in your workspace that classes should be loaded to resolve references.

  In Application Developer the path is set for the project as a whole. This is closer to a real Java environment where the VM has a classpath at run time.

- **Automatic compilation**

  Both VisualAge and Application Developer support *automatic incremental compilation*. This is normally the way you want it to work. However, if you are doing a large re-factoring of code, and you know that this will break a lot of classes, you probably want to postpone building until you have made all the changes. In VisualAge there is no way to control this, while in Application Developer you can turn of the automatic build function. See "Automatic builds" on page 30 for information on how to turn automatic builds on and off.

7

Creating HTML resources

This chapter discusses the following topics:

- Creating a Web project
- Working with Page Designer
- Creating and linking Web pages
- Creating and using a Java applet
- Creating Web pages from a JavaBean
- Importing an existing Web site
- Creating and using graphics
- CSS File Wizard

Note: The HTML and Java code that is developed and discussed in this chapter is included in the sample code that can be downloaded for this Redbook. Please see Appendix C, “Additional material” on page 675 for instructions on how to find and install the sample code.

© Copyright IBM Corp. 2002

7.1 Creating a Web project

Web projects contain all the resources needed to create a Web application, including servlets, JSPs, JavaBeans, static documents (for example HTML pages or images), and any associated metadata. This chapter will mainly be concerned with static HTML pages, but dynamic resources like JSPs and servlets will also be introduced. These types of resources are discussed in more detail in later chapters.

To create a new Web project select **File—>New—>Project.** In the wizard selection window, Figure 7-1, select **Web** and **Web Project**.

Tip: If you are already in a Web project you can bypass the Wizard Selection Window by selecting **File—>Project—>Web Project**

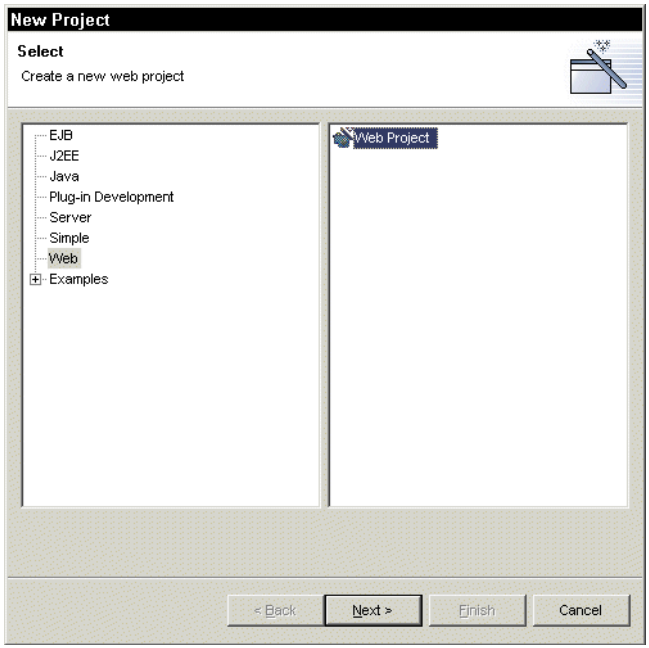


Figure 7-1 Selecting the Web wizard

Clicking **Next** will bring up the Web project wizard (Figure 7-2).

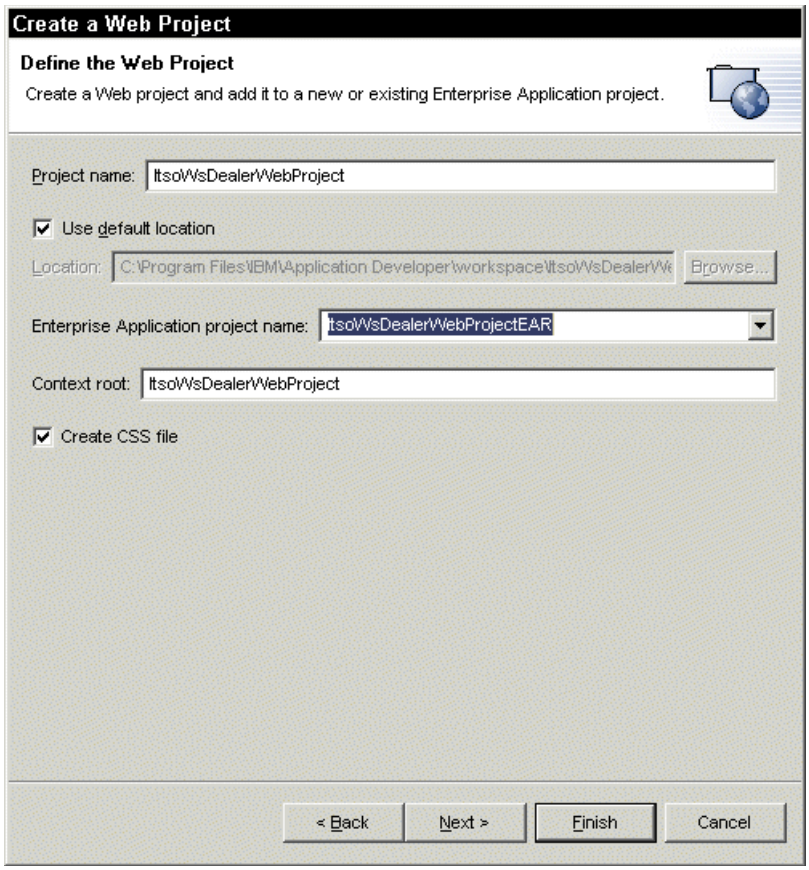


Figure 7-2 Define Web project name and location

On the first page of the Web project wizard you specify the name of the new Web project, and the location of the project files. You also need to specify the name of the Enterprise Application project that will be created along with the Web project. See “J2EE architecture” on page 112 for a description of EAR files. You have the choice of adding the project to an existing EAR or to create a new one.

The *Context root* of the project is also specified here. This is the top level directory for your application when it is deployed to a Web server. The default is to use the project name as the root. If you later on want to change this, it can be done via the project properties.

Finally you can request the wizard to create a default *Cascading Style Sheet*, (CSS), that will be applied to pages within the project. If you select this option, a file named Master.css will be created in a subfolder named *theme*.

On the next page you can add any dependent JAR files that exist in the EAR (Figure 7-3). This page will only have entries if you are adding your new Web project to an existing EAR containing JAR files.

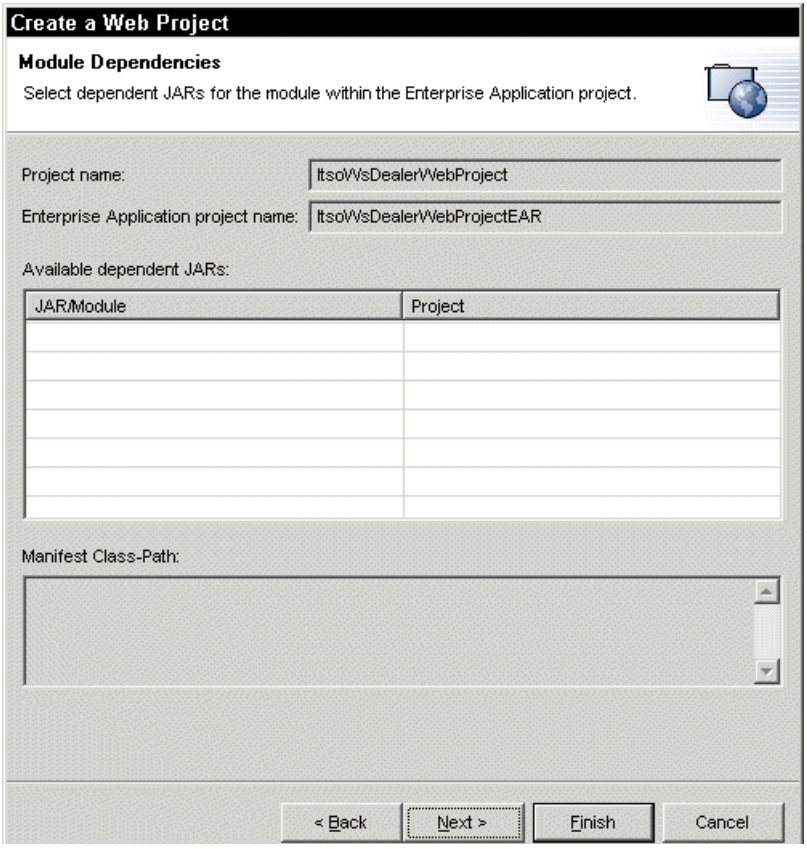


Figure 7-3 Add dependent JARs

On the final page of the wizard you can specify Java Build settings for the Web project (Figure 7-4). This page is identical to the page used when defining standalone Java projects. For a description of the various tabs see “Java applications” on page 140.

Click **Finish** to start creating the project. The wizard will create all the required directories for the project.

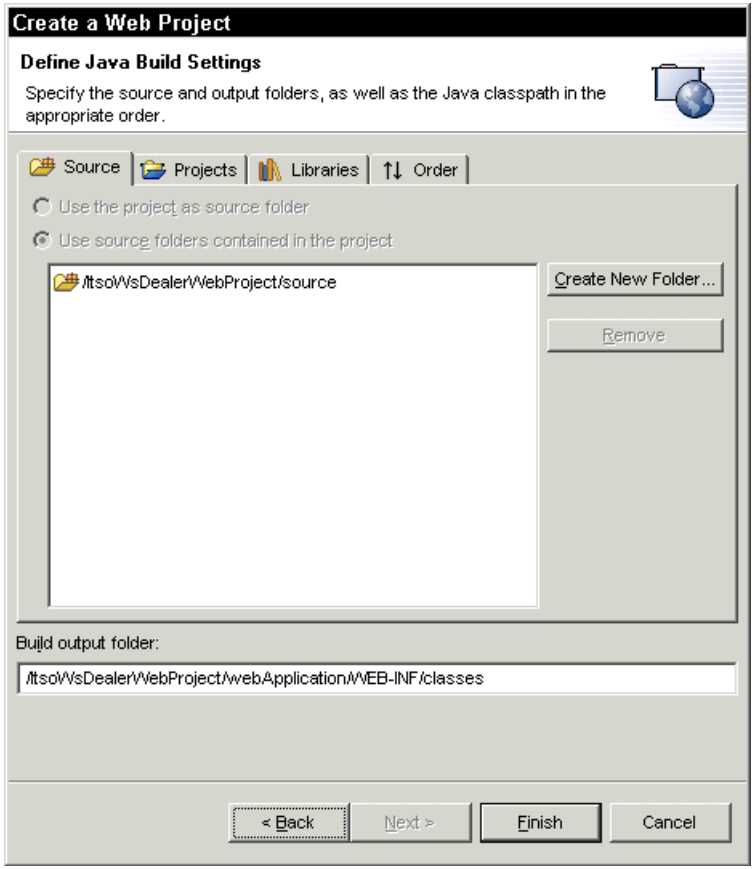


Figure 7-4 Java build settings for Web project

7.1.1 Web project directory structure

The Web project uses a J2EE directory structure in creating the following folders:

- **source:** Contains the project Java source code for classes, JavaBeans and servlets. When resources are added to a Web project, they are automatically compiled and the generated files are added to the webApplication/WEB-INF/classes directory. By default the contents of the source directory are not packaged in WAR files.
- **webApplication:** Holds the contents of the WAR file that will be deployed to the server. It contains all the Web resources, including HTML files, JSPs, and graphics needed for the application.

Important: Any files *not* under webApplication are considered development resources (for example .java and .sql files) and will not be deployed when the project is unit tested or published. Make sure that you place everything that should be published under the /webApplication directory.

- **webApplication/theme:** Contains cascading style sheets and other style-related objects.
- **webApplication/WEB-INF:** Contains the supporting Web resources for a Web application, including the web.xml deployment descriptor file and the classes and lib directories.
- **webApplication/WEB-INF/classes**: Contains servlets, utility classes, and the Java compiler output directory. The classes in this directory are used by the application class loader to load the classes. Folders in this directory will map package and class names. The .class files are automatically placed in this directory when the Java compiler compiles the source files from the source directory. Any files placed directly in this directory will be deleted by the Java compiler when it runs.
- **webApplication/WEB-INF/lib**: Contains .jar files that your Web application references. Any classes contained in these .jar files will be available for your Web application.

7.2 Working with Page Designer

Page Designer is the main tool used by a Web designer to create HTML pages and JSPs. It is a WYSIWYG editor that generates the underlying HTML/JSP code and frees the Web designer to concentrate on the visual aspect of the page rather than on details of HTML syntax. Page Designer currently supports the HTML 4.01 specification. When you save an HTML page it will automatically be validated for syntax and DTD compliance.

The Page Designer shows three views of an HTML page: *Design*, *Source* and *Preview*. In the Design view the work is done visually, while in the Source view you manually edit the HTML code. The Preview view shows what the page will look like to the user. Changes in the Design and Source views will automatically update each other as well as the Preview view. You can also preview your page using an external Web browser by selecting **Tools—>Web Browser—>Internet Explorer/Netscape** when in the Preview view.

When you are editing HTML files in the Source view you can use a *Content assist* feature similar to the one available when editing Java files. Pressing Ctrl-Space will bring up a picklist of context-appropriate selections to choose from.

To demonstrate the basic capabilities of the Page Designer we will walk through the building of a simple application consisting of two HTML pages: a title page and a linked page that displays a list of parts. The sample will demonstrate the following features:

- Creating a simple HTML page
- Creating an HTML page with a form
- Linking pages together

In subsequent chapters we discuss in more detail how to use servlets and JSPs to provide dynamic content.

Both of the HTML pages that are built in the following example are available in the sample code for this Redbook. If you have installed the code according to the instructions in Appendix C, “Additional material” on page 675 you will find the HTML examples used for this chapter in:

```
..\Chapter7\index.html
..\Chapter7\PartList.html
```

7.2.1 Create a simple HTML page

To add a new HTML page to a Web project, select the webApplication folder in your Web project and select **New—>HTML File** from its context menu. This will display a dialog where you give the page a name and a location (Figure 7-5).

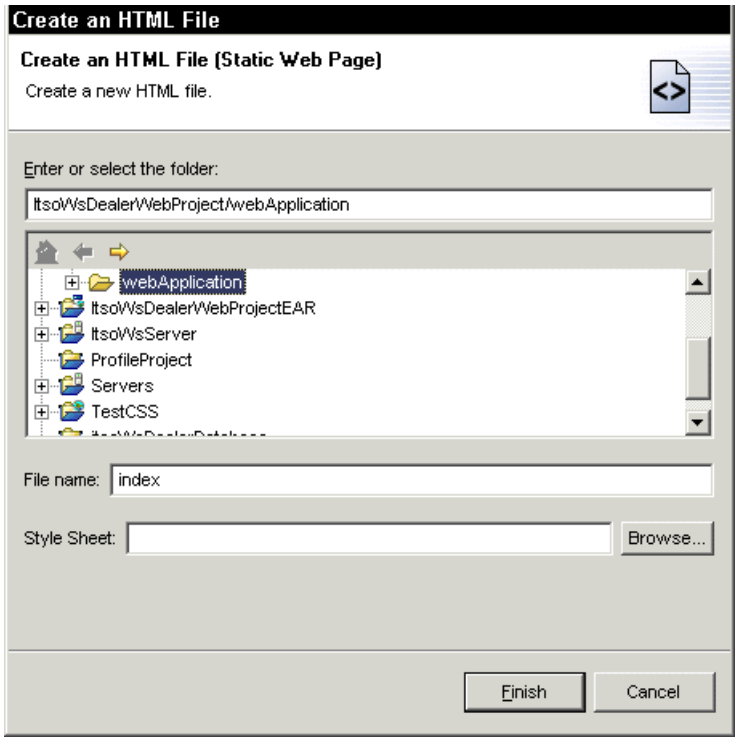


Figure 7-5 Creating an HTML page

If you had the webApplication subfolder selected, the folder field will be filled in with the correct name.

Note: HTML files can only be created in the webApplication folder or in a subfolder within it.

The only other information you have to provide is the name of the page, in this case `index`. The .html extension will be added by the wizard.

The Page Designer is opened on the new HTML page and you can now start adding your page elements. In this example we will add a page title, an image, and a static text.

To set the title for the page, display the context menu of the Design view and select **Attributes** (Figure 7-6).

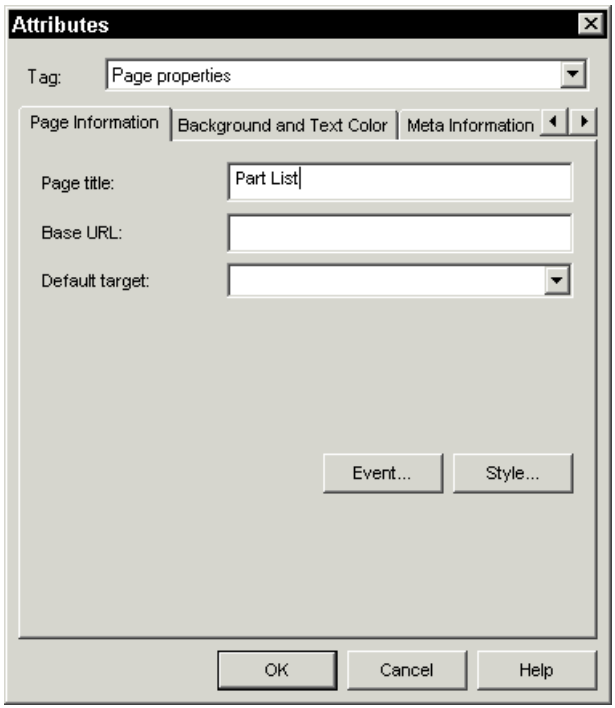


Figure 7-6 Set HTML page title

You overtype the default name, (index.html), with the title for the page (Part List) and click **OK**. If you switch over to the source tab you will note that the HTML code has been updated. You can change the `<TITLE>` tag in the source to Partlist and switch back to the Design view. Bringing up the **Attributes** dialog again will show that the title has been updated.

Next overtype the default text on the page with something more appropriate such as “WebSphere Application Programming Guide”. Finally we want to add a Redbook logo to the page.

First create a folder to contain the images. This folder must be placed under the webApplication folder to ensure that the images are included when the application is deployed to the server. Select this folder, select **New—>Folder** from the context menu and enter `images` as the folder name.

You can now import the sample image directory into the project. To do so, select **File—>Import—>File system** and navigate to `.\Chapter7\images`. Highlight images in the left view and select all the files in the directory and import them into the images subfolder (Figure 7-7).

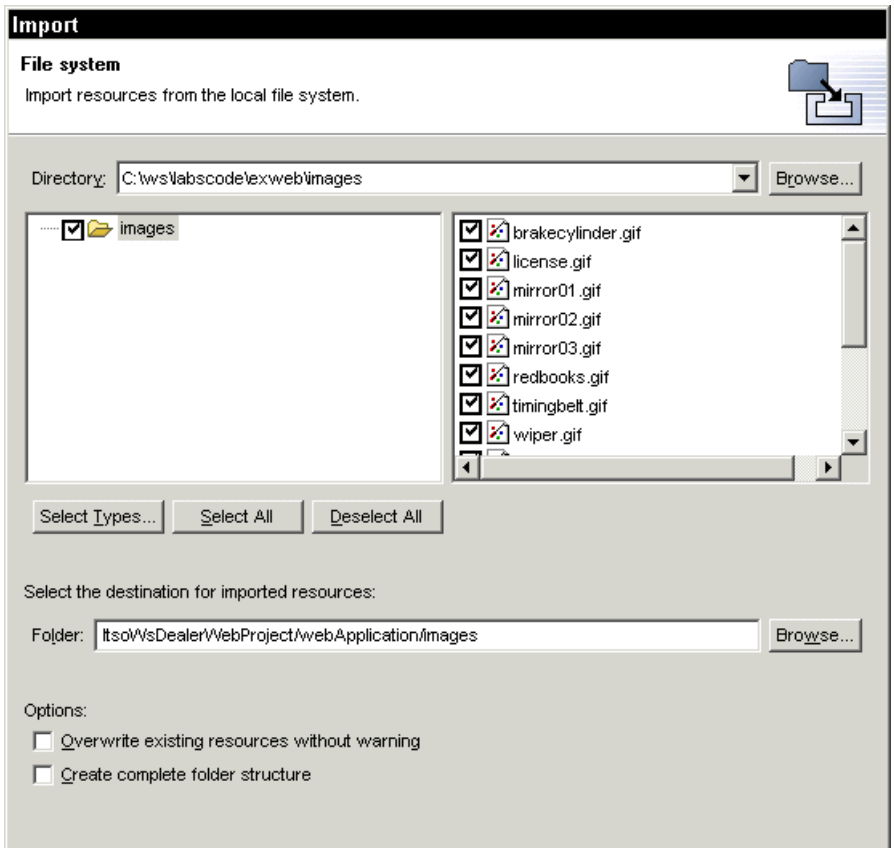


Figure 7-7 Import sample image files

Once the images are imported, select the images folder and click the Thumbnail tab in the bottom right pane. (If you can't find the Thumbnail tab, select **Perspective—>Show View—>Thumbnail** to display this view.)

You should now see thumbnail views of all the images in the image folder (Figure 7-8).

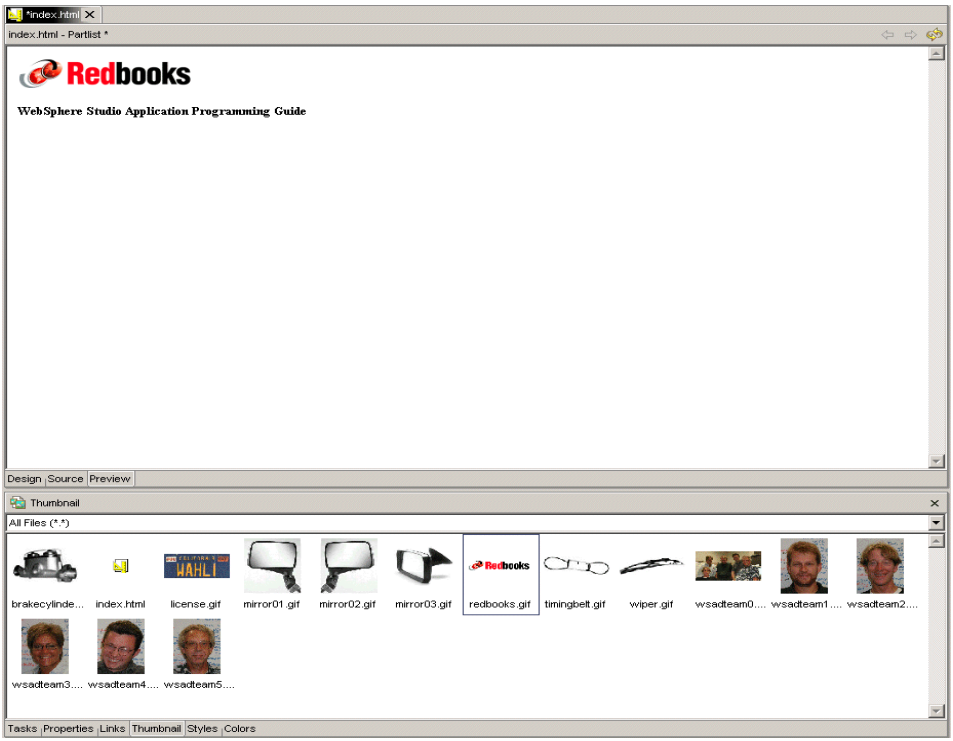


Figure 7-8 Add image to HTML page

Add the Redbook image, (`Redbooks.gif`), to the page by dragging and dropping it from the Thumbnail view. The final page should now look as in the top pane of Figure 7-8. Press `Ctrl-S` to validate and save the page.

Tip: You can change the style sheet attributes by opening the Master.css file in the theme subdirectory and selecting the **Style—>Edit** menu.

7.2.2 Creating an HTML page with a form

Now we turn to creating the second page that will invoke a servlet to return data to be displayed to the user. To create the new HTML page, `PartList.html`, you go through the same steps as previously described for index.html.

Figure 7-9 shows what the final layout of the new page will look like.

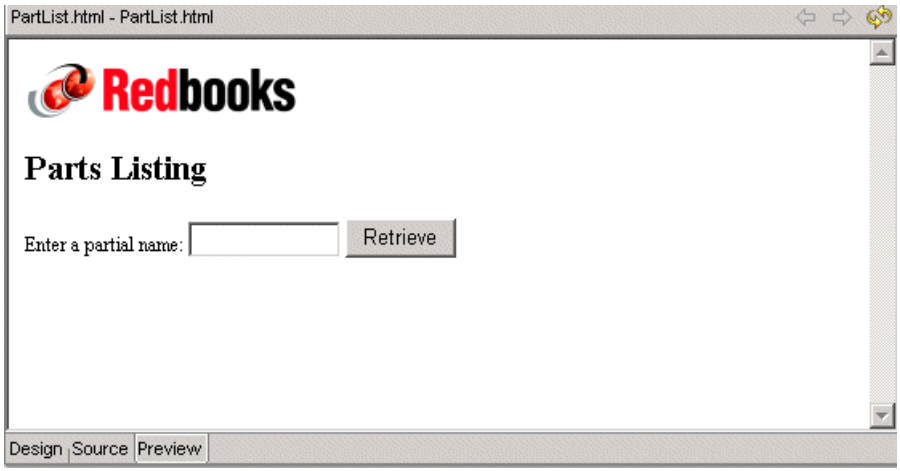


Figure 7-9 PartList HTML page

Add the page title, the static text and the Redbook logo in the same way you did for index.html.

Tip: To save some time you can make a copy of index.html, name the copy PartList.html and change the static text on the page to `Part Listing`.

Now we want to add a form field to the page. A form field allows you to accept some input from the user and perform some action on that input. To insert a form field select **Insert—>Form and Input Fields—>Form** or use the shortcut `Ctrl-0`.

Select the form and bring up the Attributes dialog from the context menu (Figure 7-10).

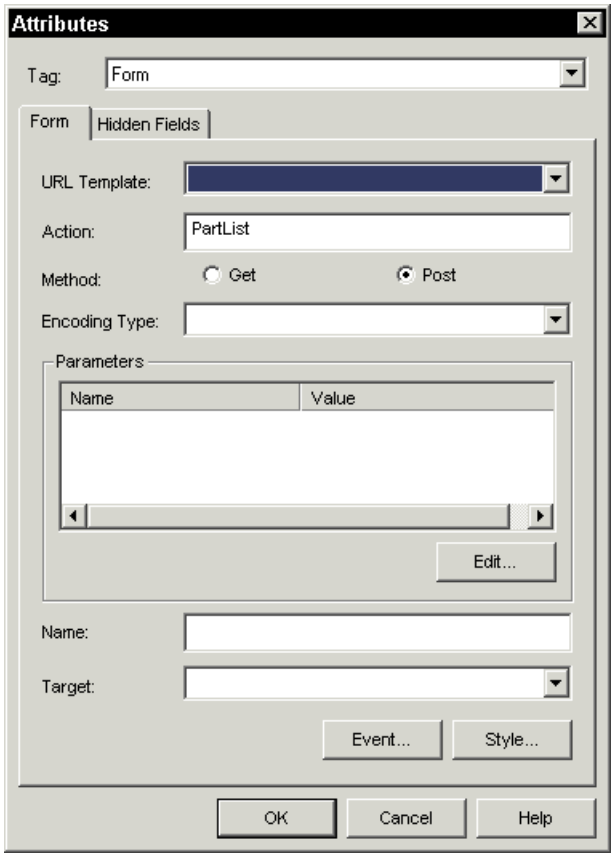


Figure 7-10 Form attributes

On this page you enter the name of the servlet to be executed when the form is submitted (PartList) in the **Action** field and specify that you want to use the HTTP **Post** method to invoke it. The PartList servlet will be discussed in detail in “Creating a simple servlet” on page 207.

The Attributes dialog generates an HTML FORM tag. In our case the action is to run a servlet, but it could also be a URL pointing to a CGI script or a mailto. In this dialog you can also enter parameters to be passed to the target of the action, and how the next page should be displayed. You can also specify actions to be performed on various events relating to the form using the **Events...** push button, and apply styles to the page using **Style**. Press **OK** to close the Attributes dialog.

Next we add some user interface elements to the form:

Type in the text `Enter a partial name` at the beginning of the Form field.

Position the cursor after the text you just typed in and select **Insert—>Form and Input Fields—>Text Field**. The attribute dialog for the text field will be displayed (Figure 7-11). Enter the name for the field, `partialName`, and leave the rest of the fields as default.

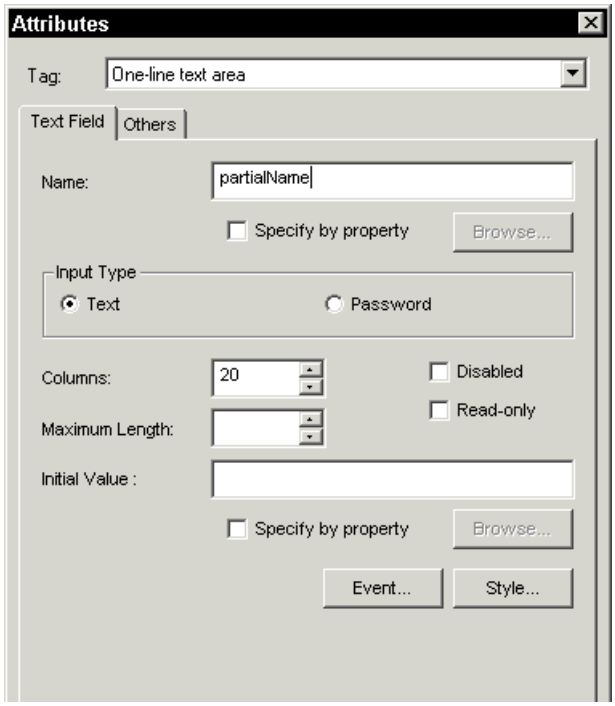


Figure 7-11 Text field attributes

Finally we need a button to invoke the action on the form. Position the cursor after the text field and select **Insert—>Form and Input Fields—>Submit Button**. The attributes dialog will be displayed (Figure 7-12). Enter a name for the button and the label to be displayed on it.

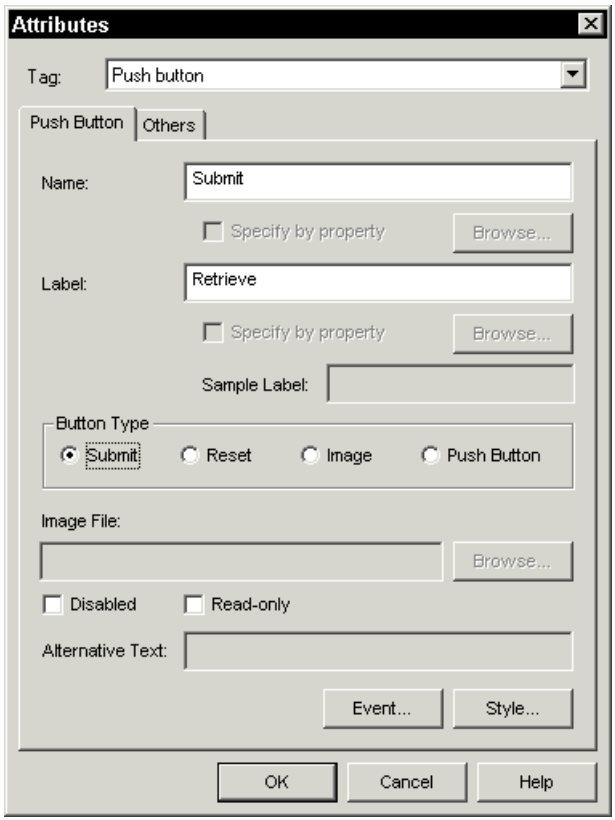


Figure 7-12 Button attributes

The page should now look as in Figure 7-9. Press Ctrl-S to save the page.

We have now created both the pages that constitute our simple Web application. Next we look at how to link them together.

7.2.3 Linking to another HTML page

We will add a link from the first page, (index.html), to the second page (PartList.html). The easiest way to add a normal link is through the Link insertion wizard. From the Design view of the index.html file, select **Insert—>Link Insertion Wizard** from the menu bar (Figure 7-13).

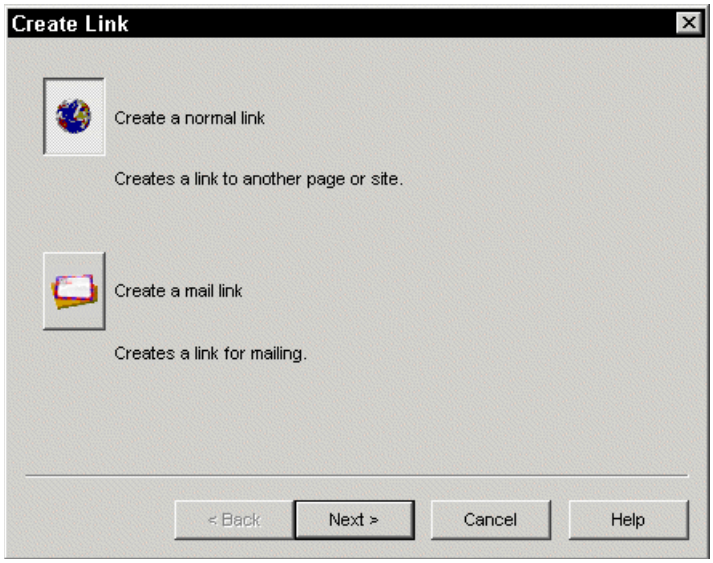


Figure 7-13 Link insertion wizard (page 1)

Click **Create a normal link** icon and then click **Next**. A normal link is a link to a URL. The other button on this page would create a *mail link*, that would invoke the user's default e-mail application to send a note to a specified e-mail address.

On the second page of the wizard, you specify the file name and the label to be shown on the link tag on the page (Figure 7-14). You can click **Select** to bring up a menu showing possible sources of the link. For our example we are linking to a file in the same directory (`PartList.html`). This file can be located by selecting **Files** from the menu. You could also create a link to a URL or to a file in another project.

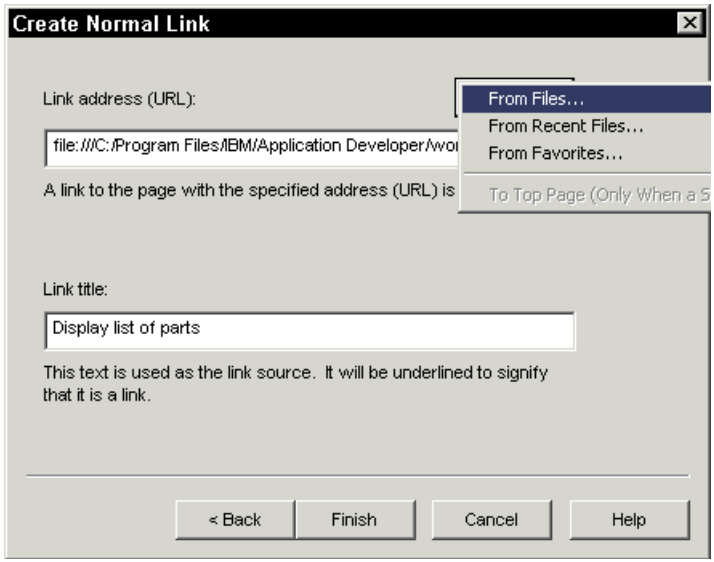

Create Normal Link
Link address (URL):
From Files...
From Recent Files...
From Favorites...
file:///C:/Program Files/IBM/Application Developer/wo
A link to the page with the specified address (URL) is
To Top Page (Only When a S
Link title:
Display list of parts
This text is used as the link source. It will be underlined to signify that it is a link.
< Back
Finish
Cancel
Help

Figure 7-14 Link insertion wizard (page 2)

Select PartList.html from the list of files and enter the link label that will be displayed on the page. Then click **Finish** to create the link in the title page. It will be displayed in the default blue color with an underline. To test the link you can go to the Preview tab and click on it. The PartList page should be displayed. Notice that the link changes color to indicate that this is now a visited link.

You can use **Format—>Align** and **Format—>Indent** to position the link on the page.

Go to the Preview view to see the finished look of the page. If you click the **Retrieve** button from this view an HTTP error page will be displayed. This is because the servlet that is invoked, and the JSP that it uses need to execute on a server. In Chapter 8, “Creating Web applications with dynamic content” on page 203 we describe how all the parts hang together and how you can run the finished application.

7.3 Creating and using a Java applet

A Java applet is a special type of Java class that executes in the context of a Web browser. There are several sources where you can find information about applets and how to write them. The following URL has a tutorial that walks you through the development process and shows several examples of applets:

`http://java.sun.com/docs/books/tutorial/applet/index.html`

Applets are visual components that usually contain graphics and UI controls.

Note: Currently there is no equivalent of the VisualAge for Java Visual Composition Editor, (VCE), for developing visual classes in Application Developer.

It is of course possible to use Application Developer to write your applet code as you would any other Java code. However you will then need to export the resulting class file and other resources and import them back into your Web application. A common scenario could be that you have an existing applet that you want to use in your application. Such existing code can be imported and used in your Application Developer Web project.

To demonstrate how to use a Java applet in a Web page we will step through building and using a class that performs the same function as the **Listing** class developed in “Java applications” on page 140. The applet code is imported into Application Developer and embedded into an HTML page.

The source code for the PartListApplet class is very similar to the Listing class. The differences are that PartListApplet inherits from JApplet rather than Object. Also the System.out.println calls have been replaced with code to create and populate a Swing table.

The source for the applet is included in the sample redbook code. You can find it in:

```
..\Chapter7\PartListApplet.java
```

Once the applet code has been written and debugged it can be imported into Application Developer. To perform the import select **File—>Import**. Then select **File system** as the source of the import and click **Next** to display the import dialog (Figure 7-15).

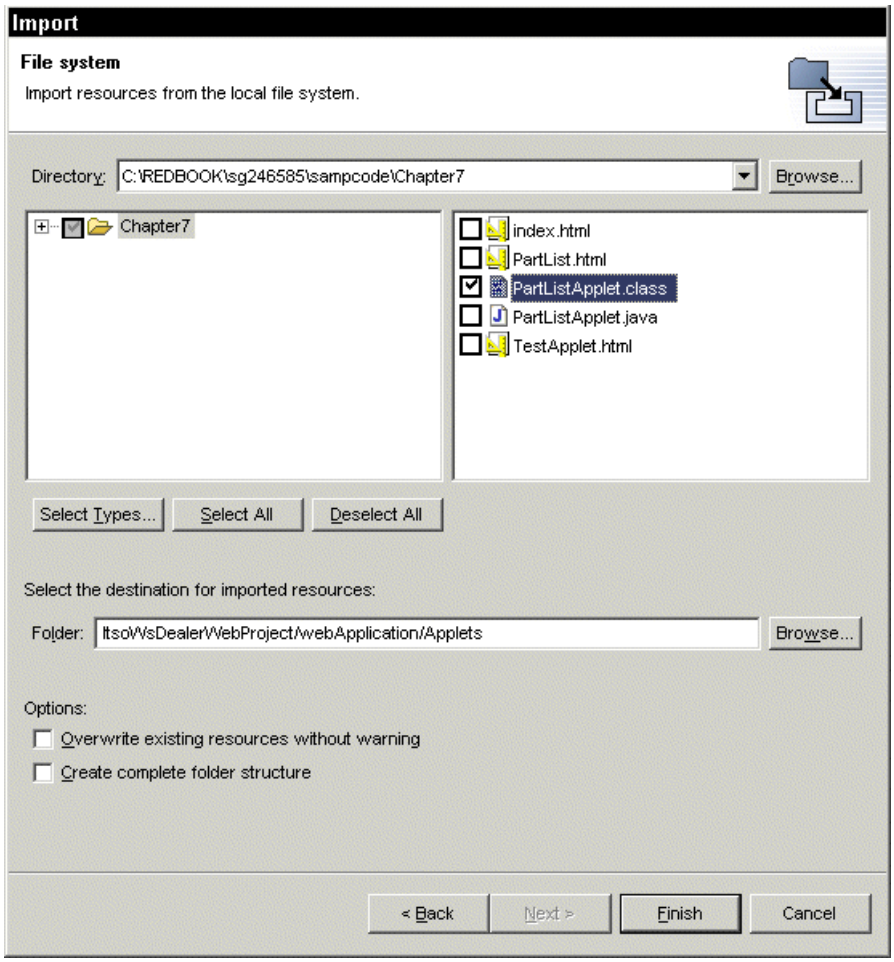


Figure 7-15 Import applet class from file system

Use **Browse** to locate the ..\Chapter7 subdirectory of the Redbook sample code, and select PartListApplet.class file. As the destination, select a folder within the Web project where you want to use the applet. For this example a folder named *Applets* has been created under ItsoWsDealerWebProject/webApplication. Clicking **Finish** will import the class code.

Tip: If you haven't created the subdirectory where you want the class stored before staring the import, you can do so during the import dialog by typing in the full folder name. Application Developer will create the folder for you if it doesn't exist.

Since the applet uses the Java Swing UI classes you also need to ensure that you have access to the JAR file containing these classes. One way of doing this is to import the swingall.jar file into the same directory as the applet code. You will find a copy of the swingall.jar file in your DB2 Java directory, for example `.\Program Files\SQLLIB\java`. Repeat the import process for this file.

The next step is to create the Web page in which to embed the applet.

Tip: You can create the page according to the following instructions or you can import the sample page TestApplet.html from `..\Chapter7` into the /webApplication folder.

The steps for creating a Web page are described previously in “Create a simple HTML page” on page 167. Select the new page and make sure the Design tab is displayed. Position the cursor where you want the applet result to appear and select **Insert—>Others—>Java Applet** from the menu bar. This will bring up the Applet dialog (Figure 7-16).

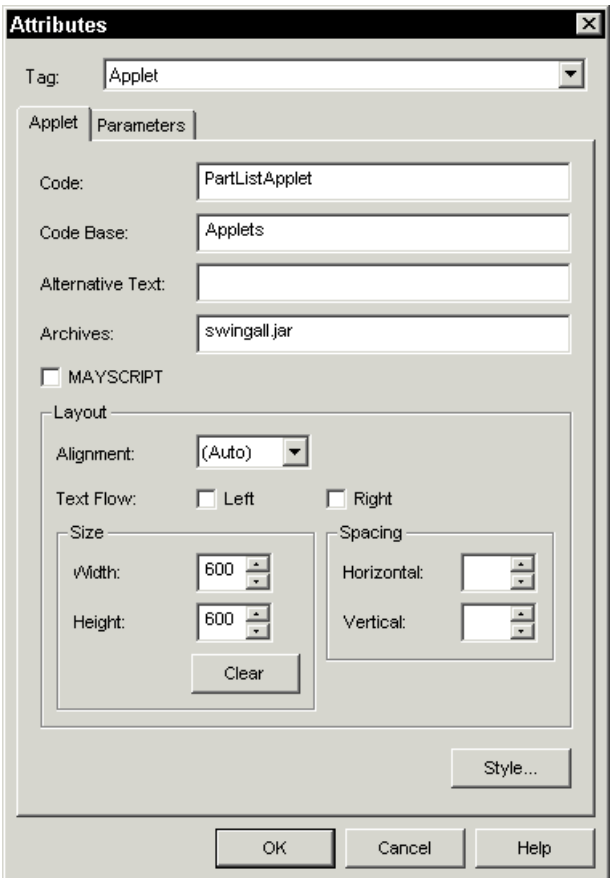


Figure 7-16 Define applet attributes

Enter the name of the applet in the **Code** field. In the **Code Base** field enter the name of the subfolder into which you imported the code. In the **Archives** field enter `swingall.jar`, (the file imported to give access to the Swing UI classes). Then choose a width and height for your applet and click **OK**.

In the Page Designer you will see a placeholder for the applet. To see the actual output from the applet, select the Preview tab. Sample output from PartListApplet is shown in Figure 7-17.

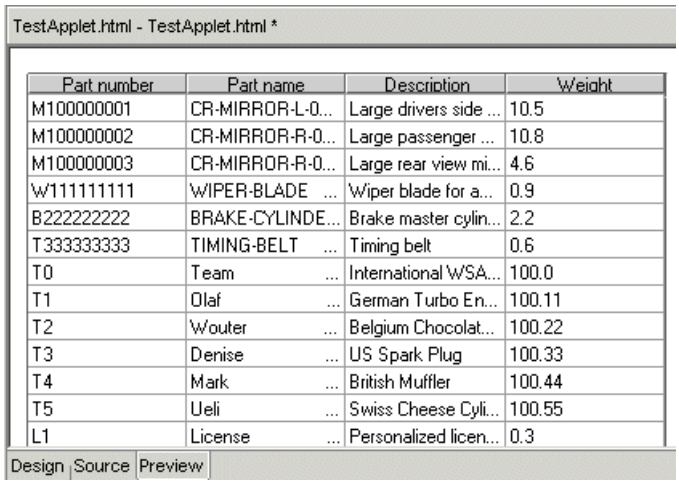


Figure 7-17 Sample applet output

7.4 Creating Web pages from a JavaBean

Application Developer provides support for generating a working set of Web pages and supporting Java classes from a bean. A JavaBean in this context, is any Java class that has a public constructor and public getters and setters for its properties. Normally a bean would perform some business- or data logic, such as accessing a database. The wizard will generate the infrastructure pages and Java classes to support the Model-View-Controller pattern for interaction between the various components. This pattern is described in “Model-View-Controller pattern” on page 218.

The Creating JavaBeans Web Pages wizard supports the following activity models:

- **Set bean properties:** Create an input form that collects input from users and stores the resulting data within the properties of a JavaBean.
- **Retrieve bean properties:** Create a result form that displays a bean's current property values.
- **Execute a bean's method(s):** Run any number of methods from a bean by submitting an input form
- **Execute a bean's method(s) with parameters:** Create an input form that collects data from users, and use this data as input to a bean's methods
- **Display a method's result:** Create a result page that displays the return value of a method.

- **Combination:** Create a set of pages that include any combination of the above models.

For the purposes of this discussion we assume that the bean, `PartListBean`, has already been created. See “Creating the model Java bean” on page 224 for more information about how to create it. To work through the following example you need to import the PartListBean.java code into your project. Select **Import—>File System** and locate the file which is in `.\Chapter8\itso\wsad\dealer\web` of the sample code directory. Import it into the `source` folder (Figure 7-18).

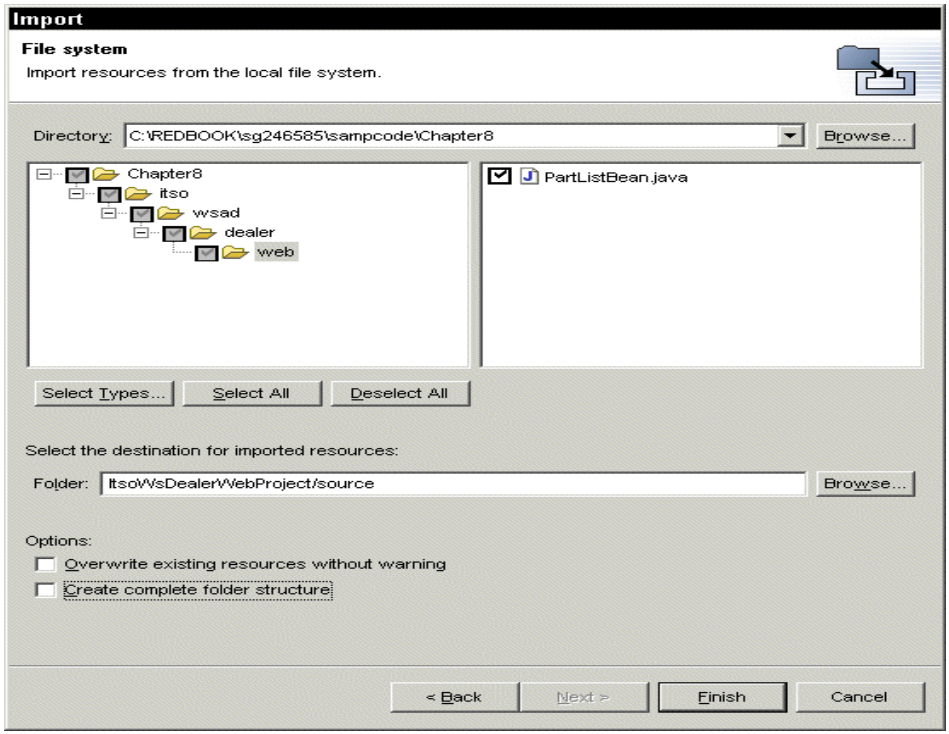


Figure 7-18 Import the bean

To generate the Web pages from this JavaBean, click the **Create Web Page from an Java bean** icon  from the toolbar.

On the first page of the wizard you specify where the generated pages and Java code should be placed and whether the results should be stored in the request or the session (Figure 7-19).

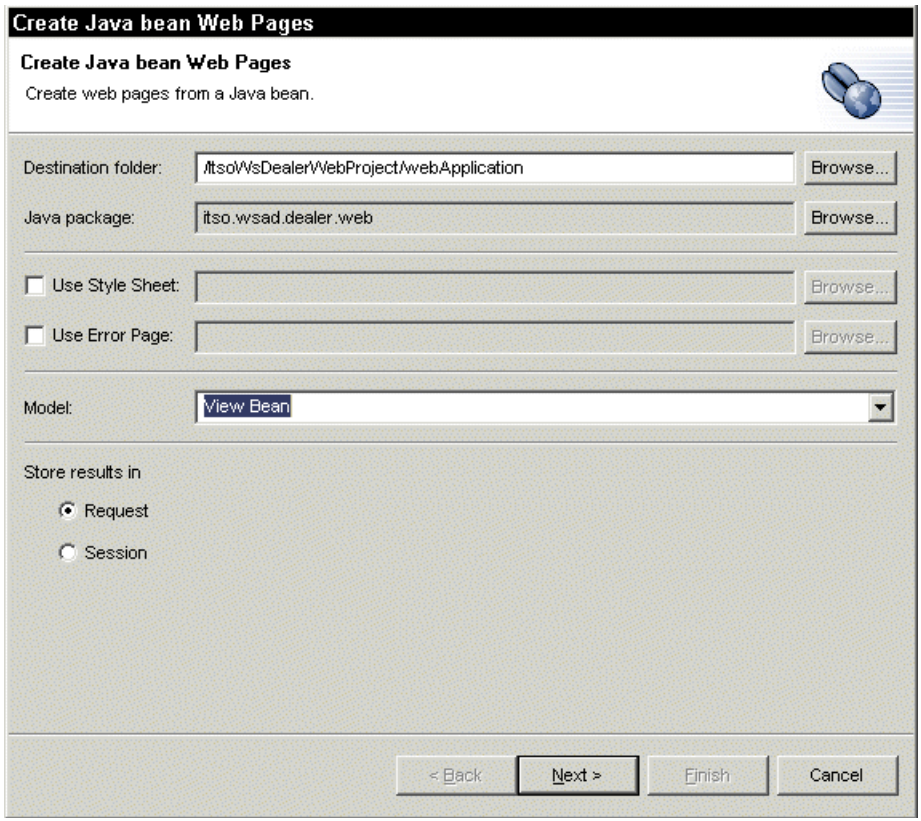


Figure 7-19 JavaBean wizard (page 1)

For our example we will store the generated pages in the default webApplication folder and servlet code in the itso.wsad.dealer.web package.

The model selection drop-down currently only has one entry: **View Bean**. This interaction model uses view helpers, which are Java-wrapper classes that manage all the database interaction.

The default for storing the results is **Request**. Using this option means that the result set will only be available for the duration of the HTTP request. If you want the result to be preserved for the lifetime of the session, choose the **Session** button. If the result set is large, memory usage may be a concern using this option.

Click **Next** to move to the next page of the wizard (Figure 7-20).

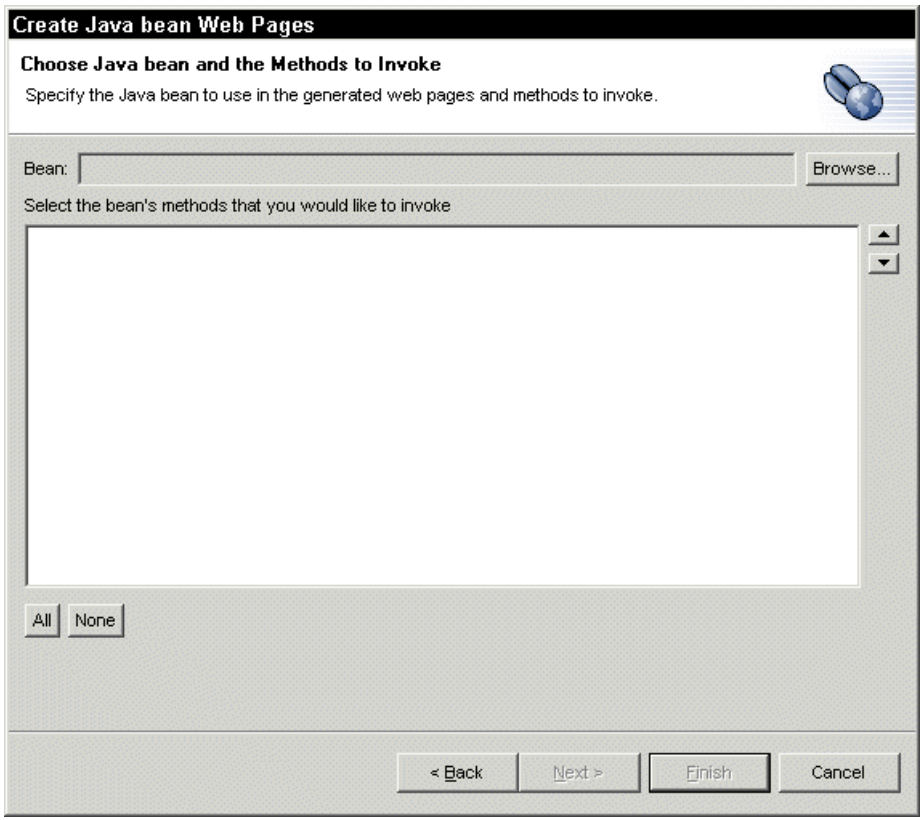

Create Java bean Web Pages
Choose Java bean and the Methods to Invoke
Specify the Java bean to use in the generated web pages and methods to invoke.
Bean:
Browse...
Select the bean's methods that you would like to invoke
All
None
< Back
Next >
Finish
Cancel

Figure 7-20 JavaBean wizard (page 2)

On this page you identify the JavaBeans that you want the wizard to generate Web pages for. Click **Browse** to display a list of candidate classes (Figure 7-21).

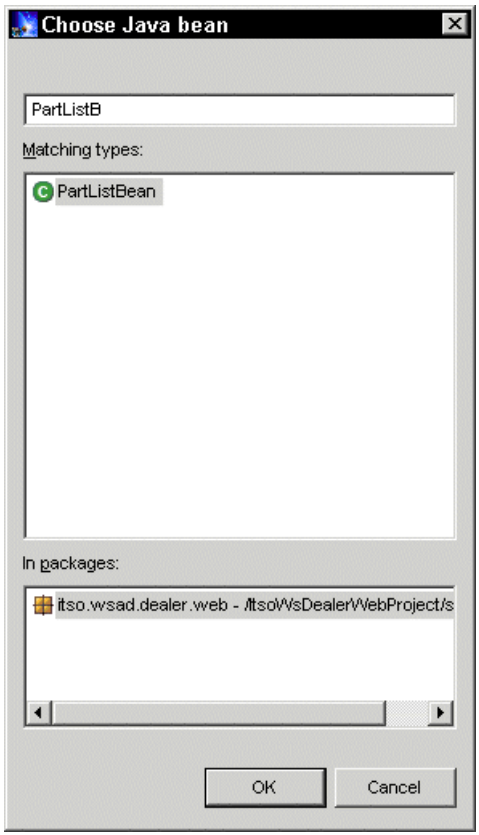


Figure 7-21 Choose JavaBean

The wizard will display all classes found in:

- The source folder of your Web project
- The JAR files included in the Java build path of your Web project
- The Java build path of your Web project

To quickly locate the class you want to use, start typing its name in the entry field until you can see the class in the **Matching types** list. Once you have found it, select it and click **OK** to return to the wizard page. That will now display a list of all the methods that you can access in the selected class (Figure 7-22).

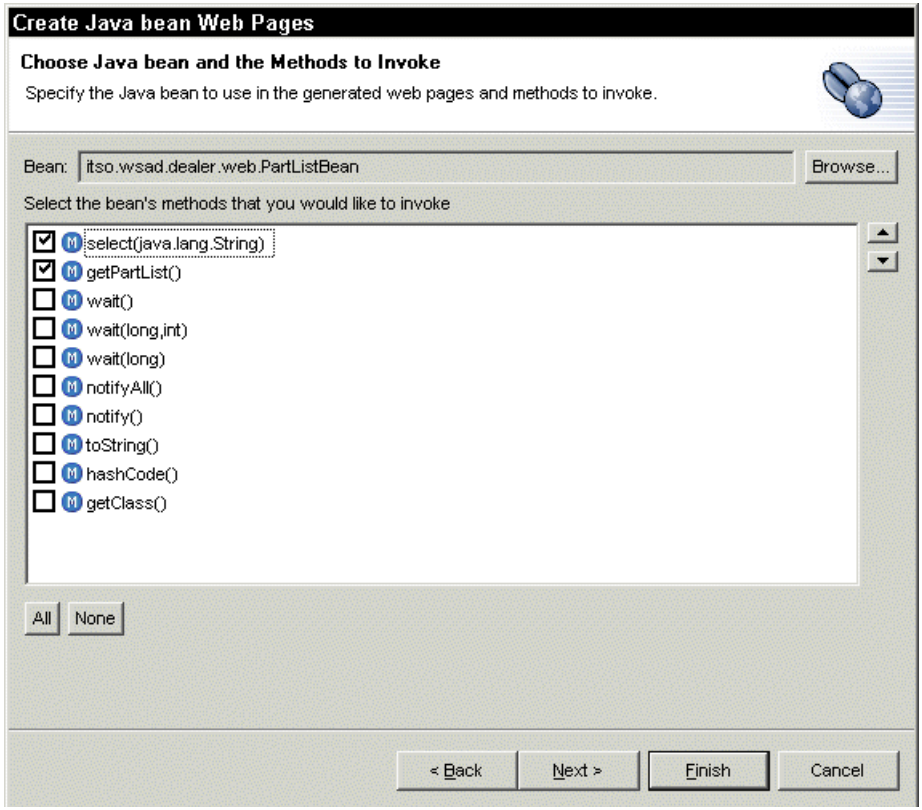


Figure 7-22 JavaBean wizard after choosing JavaBean class

The wizard will show all public methods with primitive parameter types. In this case we are interested in two methods. We want to use *select()* to query the database and *getPartList()* to return the results. Check the check boxes next to the method names.

Important: The order of the methods in the list is important. They will be called in the order they are shown here. If the methods are not in the correct order you can use the Up and Down arrow icons to move them. As shown in Figure 7-22, the select() method will be called first to retrieve the information and getPartList() will be called to display the result.

Clicking **Next** will display the page where you can make modifications to the layout of the Input form that will be generated (Figure 7-23).

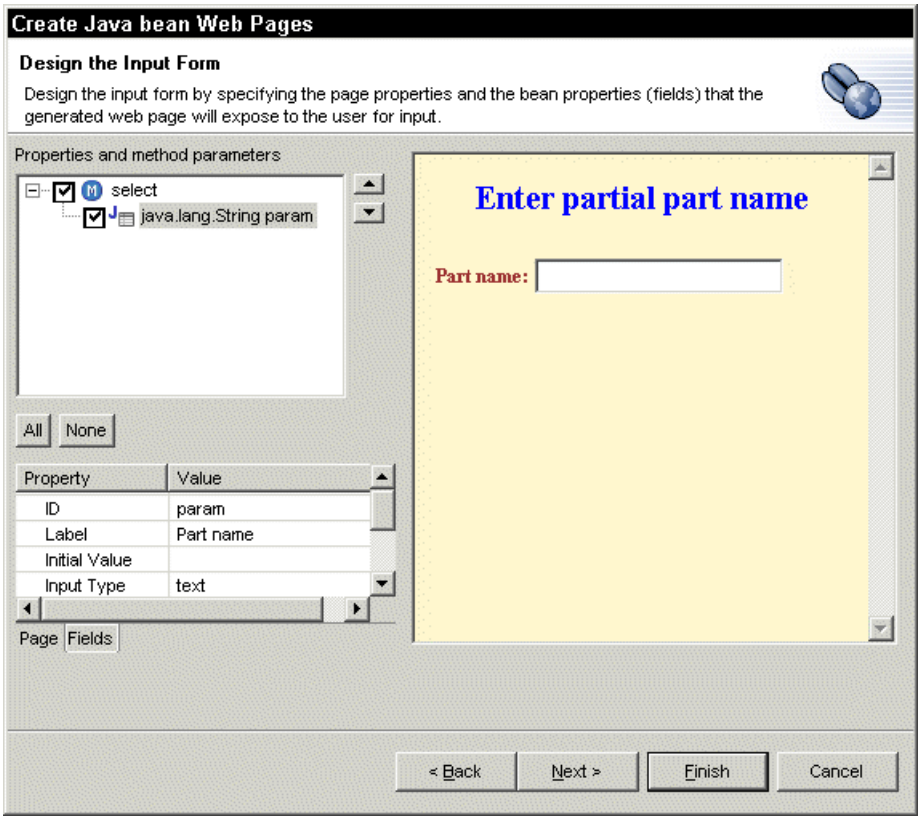


Figure 7-23 JavaBean wizard - design the input form

For this example we have modified the default title and the entry field label. (Once the pages have been generated you can of course make further changes to improve the appearance of the page.) In our example we only have one method to call on the input form. If there had been several, you can control the invocation order in the **Properties and method parameters** list.

Clicking **Next** will bring up the corresponding page for the Results (Figure 7-24).

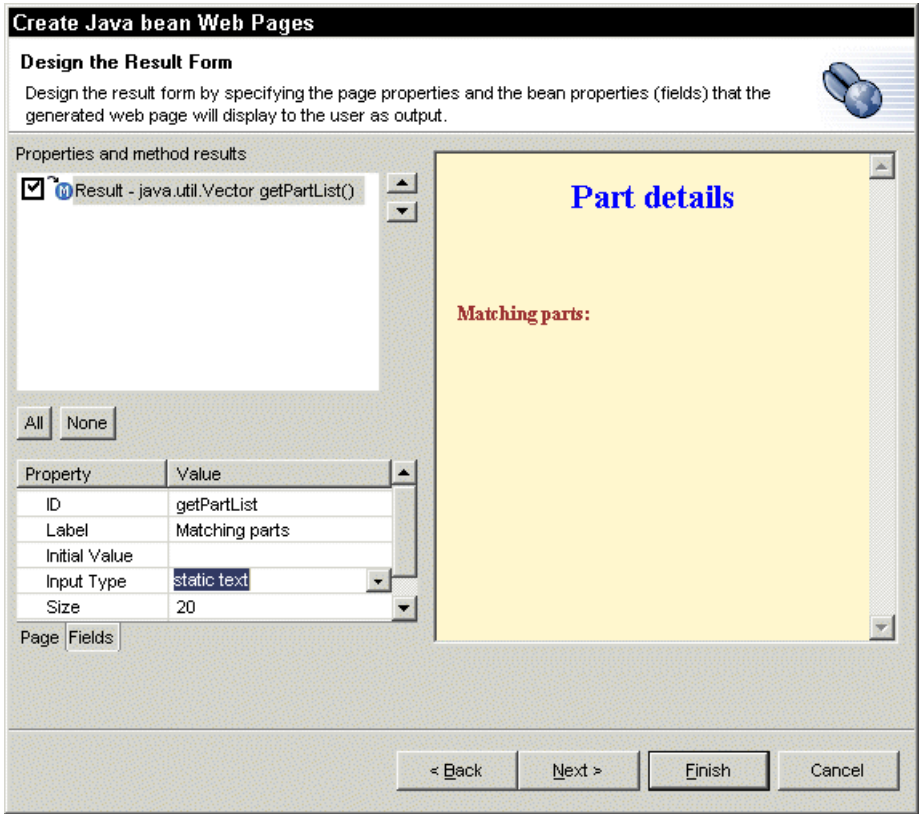


Figure 7-24 JavaBean wizard - design the result form

Again we have made some changes to the title and the label.

On the next wizard page you can modify some generating options (Figure 7-25).

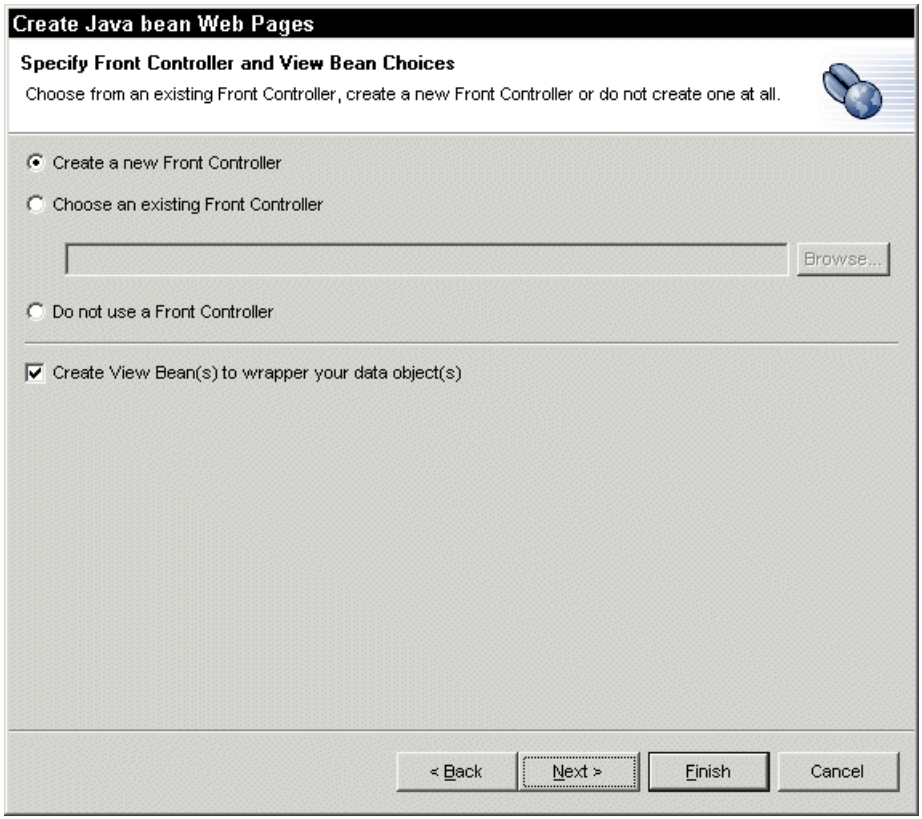

Create Java bean Web Pages
Specify Front Controller and View Bean Choices
Choose from an existing Front Controller, create a new Front Controller or do not create one at all.
Create a new Front Controller
Choose an existing Front Controller
Browse...
Do not use a Front Controller
Create View Bean(s) to wrapper your data object(s)
< Back
Next >
Finish
Cancel

Figure 7-25 JavaBean wizard - specify controller and View Bean options

By default the wizard will generate a controller servlet to invoke the bean. You can choose not to generate the controller or to use an existing one. On this page you can also de-select the creation of a View Bean wrapper. This wrapper class allows you to modify the result data for display. If you deselect this option, the wizard accesses data using the underlying beans directly.

On the final page of the wizard you can control the naming of the generated files (Figure 7-26).

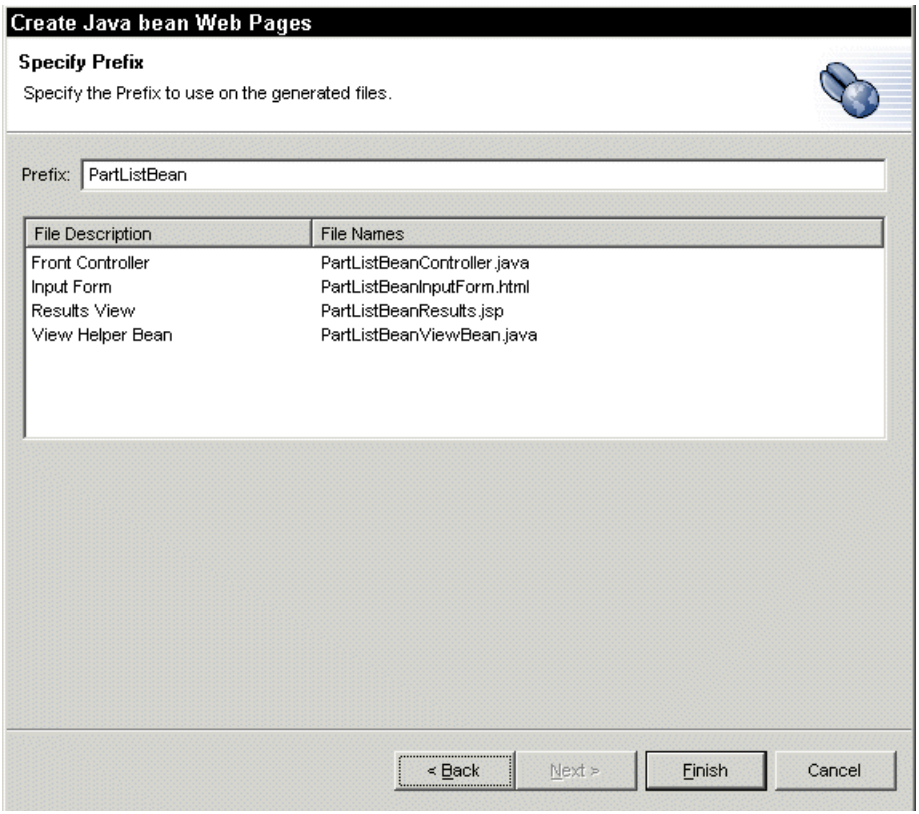


Figure 7-26 JavaBean wizard - specify prefix

Clicking **Finish** will generate the code and forms. The following files will be created in the specified output directory:

- **PartListBeanController.java**: The controller servlet.
- **PartListBeanInputForm.html**: The HTML input form to enter the parameter and submit the request.
- **PartListBeanViewBean.java**: View wrapper helper class to format the bean output.
- **PartListBeanResults.jsp**: Page to display the results.

Once the generation has finished, the Web pages can be tested by selecting PartListBeanInputForm.html in the output folder and selecting **Run on Server** from its context menu. When the input form is displayed you type in a parameter, for example `M`, and click on **Submit**. The PartListBeanResults page will be displayed showing the results retrieved by the Java bean.

Note: You will notice that the application runs quite slowly the first time. This is because Application Developer creates a server instance to run the program, starts the server, and compiles the JSP into a servlet the first time it is referenced. Subsequent executions will run significantly faster.

Since the getPartList() method in the bean returns a Java vector, and we are at the moment not doing any formatting on it, the result will not look very meaningful.

To change the way the result is formatted, you should locate the *massageOuput()* method in the View bean, (in our example PartListBeanViewBean.java). In this method you can insert any formatting code you need to convert between what has been returned from the JavaBean and what you would like to display on your result page. To get the sample result list to display in a somewhat more “user friendly” fashion you could add the following code to this method:

```
//Place code here to format your output
out = "";
java.util.Vector resultList = (java.util.Vector)in;
for (int i = 0; i < resultList.size(); i++) {
String[] resultRow = (String[])resultList.elementAt(i);
out = out + resultRow[0] +" " +
resultRow[1] +" " +
resultRow[2] +" " +
resultRow[3] +" " +
resultRow[4]+" "+"<BR>";
}
```

If you re-run the application you should now see a result looking similar to Figure 7-27.

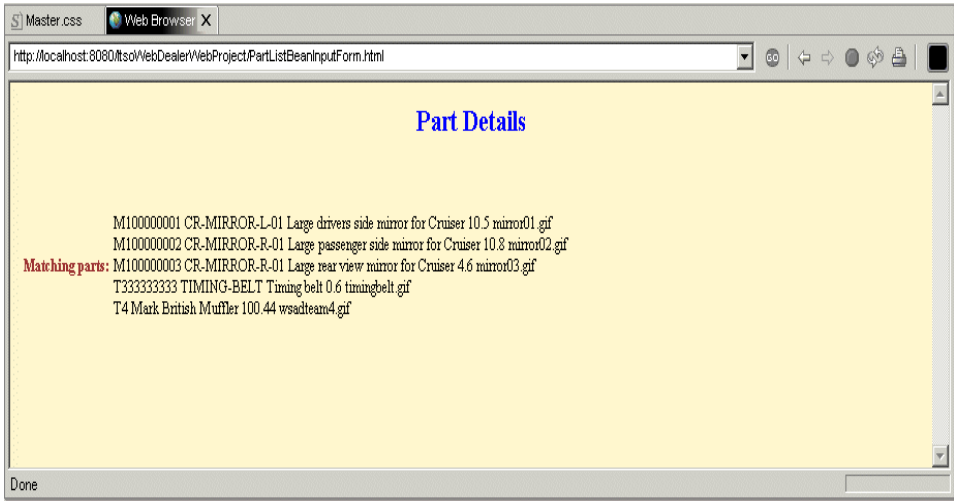

Master.css
Web Browser
http://localhost:8080/itso/WebDealer/WebProject/PartListBeanInputForm.html
Part Details
M100000001 CR-MIRROR-L-01 Large drivers side mirror for Cruiser 10.5 mirror01.gif
M100000002 CR-MIRROR-R-01 Large passenger side mirror for Cruiser 10.8 mirror02.gif
Matching parts: M100000003 CR-MIRROR-R-01 Large rear view mirror for Cruiser 4.6 mirror03.gif
T333333333 TIMING-BELT Timing belt 0.6 timingbelt.gif
T4 Mark British Muffler 100.44 wsadteam4.gif
Done

Figure 7-27 Sample output from the generated application

Once you have a working skeleton application you can proceed to make further additions and changes as required to produce the final look and feel of your pages.

In a very similar fashion to the way you create Web pages from a JavaBean, you can create them from an SQL statement. This is described in detail in “Generate Web pages from SQL queries” on page 304.

7.5 Importing an existing Web site

In Application Developer you can import all or part of an existing Web site to use in your application. You may have developed a Web site using some other tool, and you now want to continue developing and testing it using the Application Developer Web development environment.

Note: The import utility will only import static content such as html pages and graphics.

To import a Web site, highlight the folder where you want to store the imported files and select **Import->HTTP** from the menu bar. The Web site import dialog will be displayed (Figure 7-28).

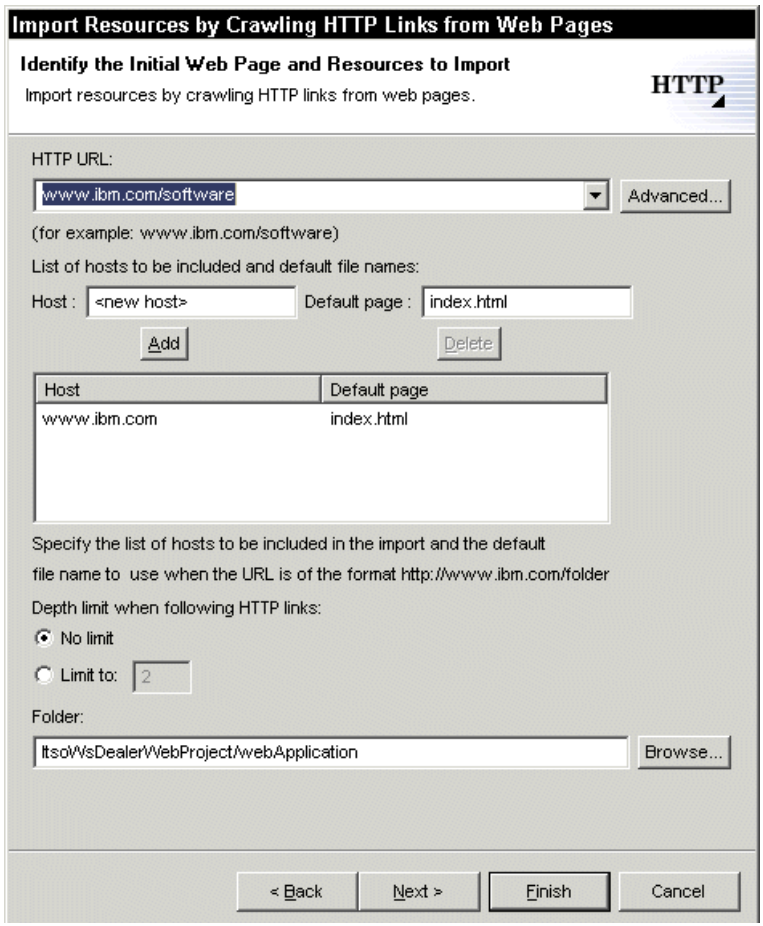


Figure 7-28 Import Web site dialog - page 1

Here you identify the URL to import from and set limits on what resources you want to include in the import. In the preceding example we are importing pages from the directory URL www.ibm.com/software. This causes www.ibm.com to be added to the list of hosts. You can add other hosts to this list if you want them to be included in the import. If you leave the default, only links within www.ibm.com will be followed. The **Default page** is used by the import process to create a start page if you're importing from a directory URL.

The **Advanced** button allow you to specify a proxy server for the URL.

Clicking **Next** will give you some more options for the import (Figure 7-29).

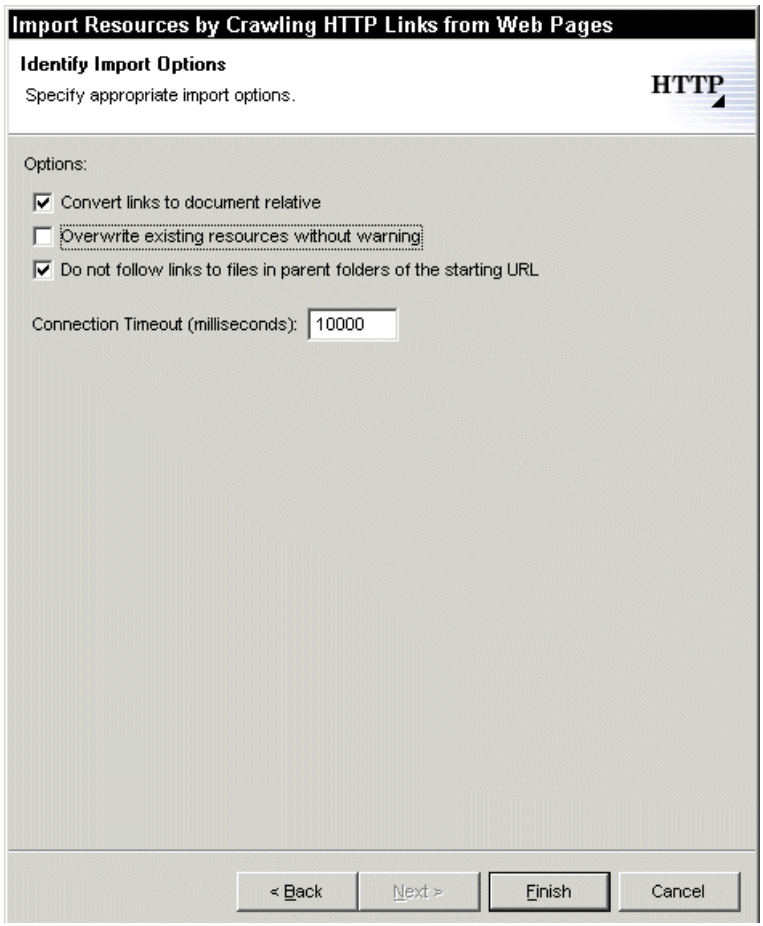


Figure 7-29 Import Web site dialog - page 2

Checking **Convert links to document relative links** causes links to be updated in a document-relative fashion, rather than creating absolute links based on their new location in the file system.

Overwrite existing resources without warning controls whether you will be prompted if an imported file will overwrite a file with the same name that already exists in the folder.

Do not follow links to files in parent folders of the starting URL determines what happens if links refer to parent folders of your selected URL. The default is *not* to follow these links. You should be careful about unchecking this option, since it could cause large amounts of files to be imported.

Clicking **Finish** will start the import. All resources from the selected URL will be created in the target folder.

7.6 Creating and using graphics

Application Developer provides a gallery of images that you can use in your Web pages. To insert an existing image make sure you are in the Design view of the Web perspective. Select **Insert—>Image File—>From Gallery**. This will bring up a window from which you can select from a large number of different types of images (Figure 7-30).

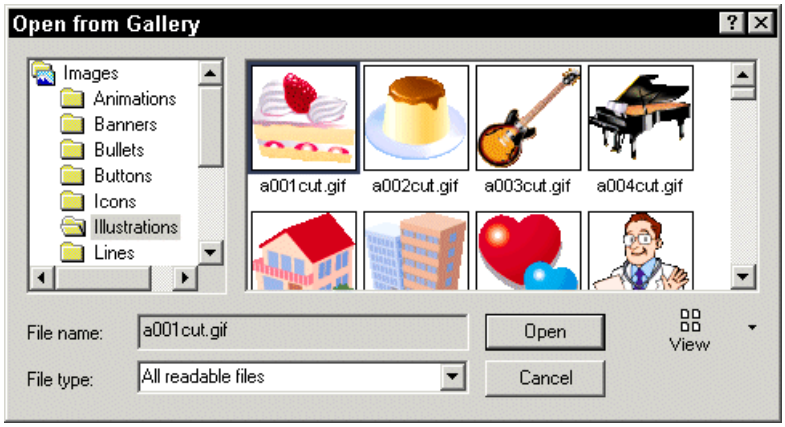


Figure 7-30 Image gallery

Other options available on the **Insert—>Image File** menu allow you to import images from the file system and from a URL. You can also drag and drop images from a Web page or from the Thumbnail view in Application Developer.

Application Developer also contains a WebArt editor that allows you to create your own image files. To access the editor, switch to the Web perspective and click the **Create an Image File** icon  from the toolbar.

In the dialog shown, Figure 7-31, you specify where the image should be placed and what graphics format it should be saved in. The following formats are supported:

- GIF
- MIF (WebArt file)
- JPEG
- PNG
- BMP

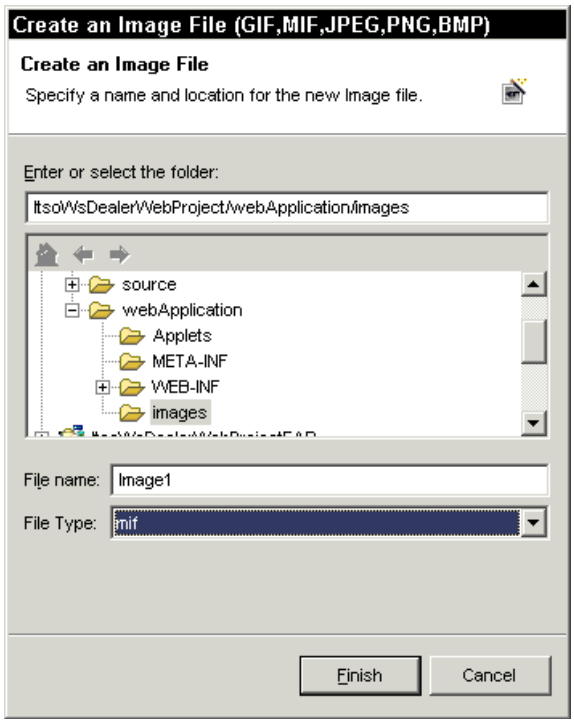


Figure 7-31 Create image file dialog

Clicking **Finish** will display the editor where you can create your image. You can also import an image created elsewhere and edit it here (Figure 7-32).

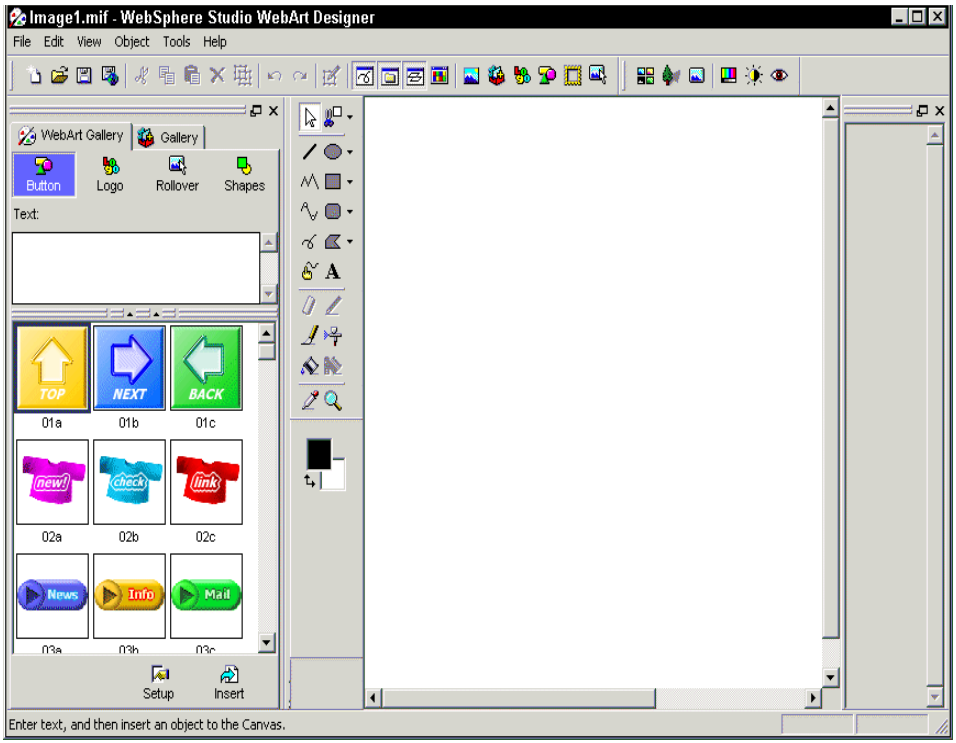


Figure 7-32 WebArt editor

All the images from the gallery shown in Figure 7-30 are also available to use from within the WebArt editor. To access them, select the **Gallery** tab in the left pane.

7.7 CSS File Wizard

A *Cascading Style Sheet*, (CSS), allows authors and readers to attach various styles, for example fonts, colors and spacing, to an HTML document. As was mentioned earlier, “Creating a Web project” on page 162, you can ask Application Developer to create a default CSS for you when you define your project. Application Developer also provides a special editor to modify CSS files or create new ones. To access the CSS editor, either double-click on an existing style sheet in your Web project, or click the **Create a CSS File** icon  in the toolbar.

This will bring up the Create a CSS File dialog (Figure 7-33), where you can specify the location of the new CSS file and its name. We recommend that you store CSS files in the WebApplication/theme folder. If you let Application Developer create a CSS file for you during project definition, you will find it in this folder as Master.css.

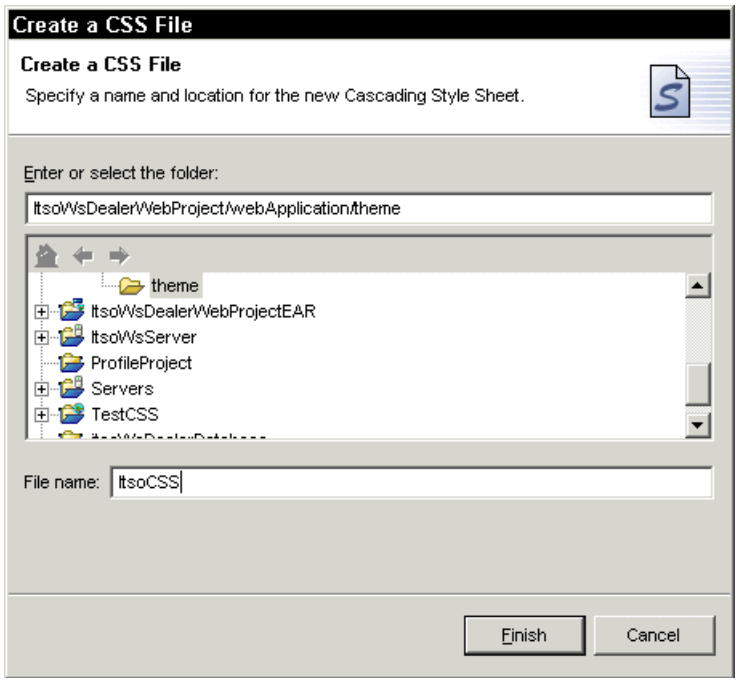


Figure 7-33 Create a CSS file dialog

Clicking **Finish** will open the CSS file editor (Figure 7-34).

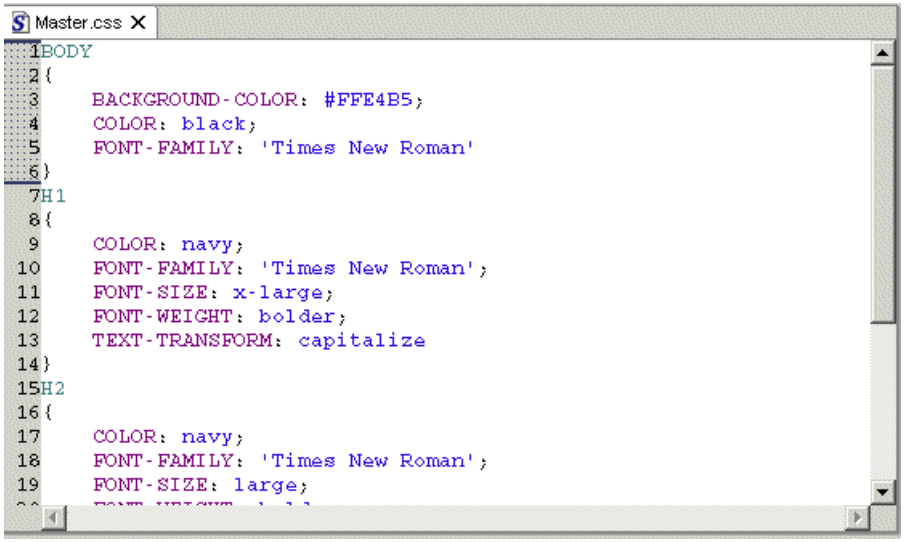


Figure 7-34 CSS file editor

If you are familiar with CSS file syntax you can edit this file directly. If you prefer, you can instead use the **Style** menu to edit or add styles. This menu is context sensitive and will open up on the style that is currently selected in the editor. Figure 7-35 shows the edit dialog displayed when the HI style was selected in the editor and **Style—>Edit** was selected from the menu.

Just like when editing HTML files you can use the Content Assist feature to help you select values. Press Ctrl-Space and a picklist of choices appropriate to where the cursor is currently positioned will be displayed.

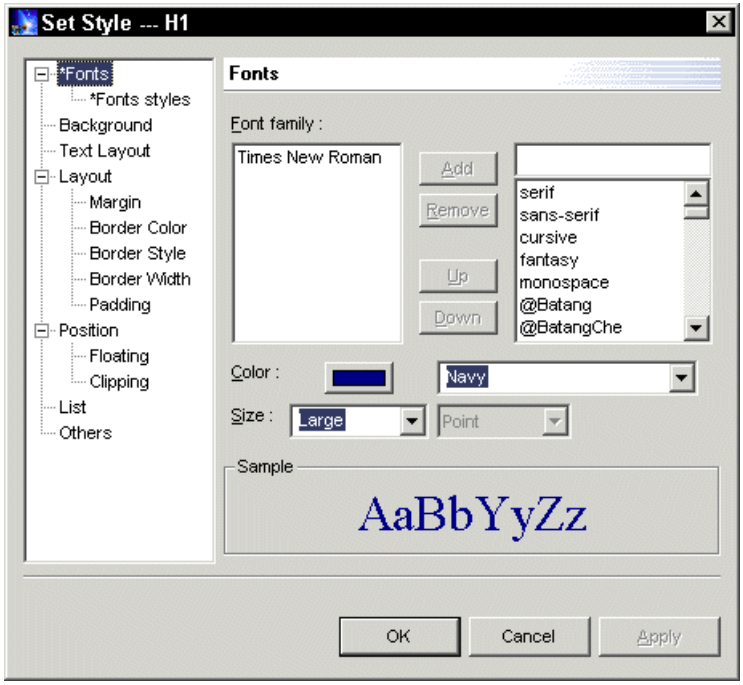


Figure 7-35 Edit style dialog

From the **Style** menu you can also add links to, and import, other style sheets.

8

Creating Web applications with dynamic content

There are many ways to create dynamic Web applications. The most time-consuming method is to build the pages manually by writing the code line-by-line in a text editor. An easier and more productive way is to use the Application Developer wizards in conjunction with content-specific editors, such as the HTML Editor, the XML Editor, and the CSS Editor. The Web development wizards help you quickly create forms, HTML pages, JavaServer Pages (JSPs), and Java servlets, even if you are not an expert programmer.

This chapter describes the following:

- Adding a servlet to a Web project
- Creating a simple servlet
- Model-View-Controller architecture
- Adding a JavaBean as a model
- Adding a JSP as a view
- Creating a controller servlet
- Testing a Web application

© Copyright IBM Corp. 2002

8.1 Wizards

The Web development wizards guide you step-by-step through the process and generate output files that you can use "as is" or modify for your Web site to provide server-side processing for dynamic content. The resulting pages will run on the WebSphere Application Server Version 4.0, Apache Tomcat Version 3.2, and any other Web server that supports the Sun Microsystems Java Servlet 2.2 Specification.

Application Developer wizards not only support you in creating SQL statements, Servlets and JavaBeans, but they also compile the Java code and store the .class files in the correct folders for publishing to the WebSphere Application Server. The wizards use the Java builder that comes with Application Developer. In addition, as the wizards generate project resources, the deployment descriptor file, *web.xml,* is updated with the appropriate configuration information for the JSPs and servlets that are created. You can test the resulting project resources within the Application Developer using the Websphere Unit Test Server Environment or the packaged version of the Tomcat server.

Note: The code that is developed and discussed in this chapter is included in the sample code that can be downloaded for this redbook. See Appendix C, “Additional material” on page 675 for instructions on how install the sample code on your computer.

8.2 Working with servlets

Servlets are server-side Java programs that use the *Sun Microsystems Java Servlet API* and its associated classes and methods, as defined in the *Sun Microsystems Java Servlet 2.2 Specification*.

Servlets extend the functionality of a Web server by generating dynamic content and responding to Web client requests. When a browser sends a request to the server, the server can send the request information to a servlet, which in turn will build the response HTML that is returned back to the browser.

Just as applets run on a Web browser to extend the browser's capabilities, servlets run on a Java-enabled Web server, such as the WebSphere Application Server, to extend the application server's capabilities. Servlets are commonly used by businesses to connect their Web applications to back-end datastores. They are flexible and scalable and the Application Developer environment provides the necessary features to make them easy to develop and integrate into your Web application.

You can develop, debug, and deploy servlets within Application Developer. In the workbench, you can set breakpoints within servlet objects, and step through the code. You can make changes that are dynamically folded into the running servlet on a running server, without having to restart the server each time.

8.2.1 Create or open your Web project

When working with JSPs and servlets you will add additional resources to a Web project. If you created a Web project in the previous chapter, you can continue to use that for the examples in this chapter.

If you haven't yet created a Web project, you may want to create one now. The steps to follow when creating a new Web project are described in detail in "Creating a Web project" on page 162. In the following sections we will assume that a project called *ItsoWsDealerWebProject* exists.

8.2.2 Adding a servlet to your Web project

Application Developer provides a **Servlet wizard** to assist you in adding servlets to your Web applications. The Servlet wizard walks you through the process of creating a Java servlet and creates output files you can use "as is" or modify for inclusion in your Web application.

The resulting servlets can run on the WebSphere Application Server or on any other J2EE-compliant Web server, and provide server-side processing for dynamic content.

To add a servlet, do the following:

Launch the Servlet wizard, selecting **File—>New—>Other** (or click the **Open The New Wizard** icon  in the top left corner of the workbench's workarea). Then select the **Web** option, **Servlet**, and then click the **Next** button (Figure 8-1).

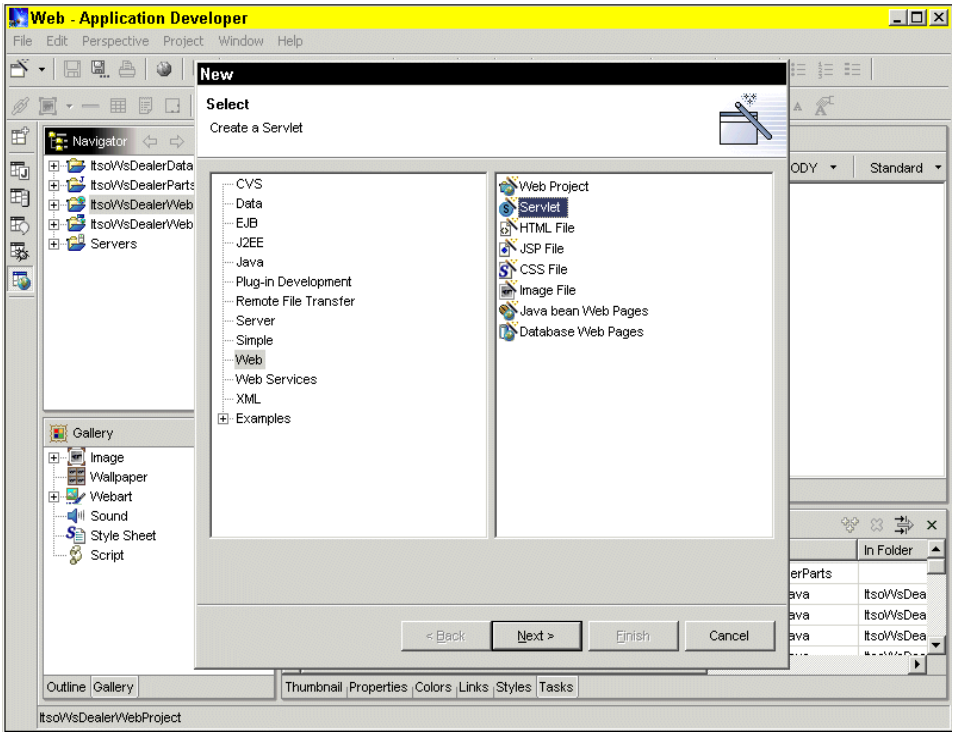


Figure 8-1 Adding servlet

If you are in the Web perspective, you can launch the Servlet wizard by selecting **File—>New—>Servlet** or by selecting the **New Servlet Wizard** icon  in the tool bar, directly (Figure 8-2).

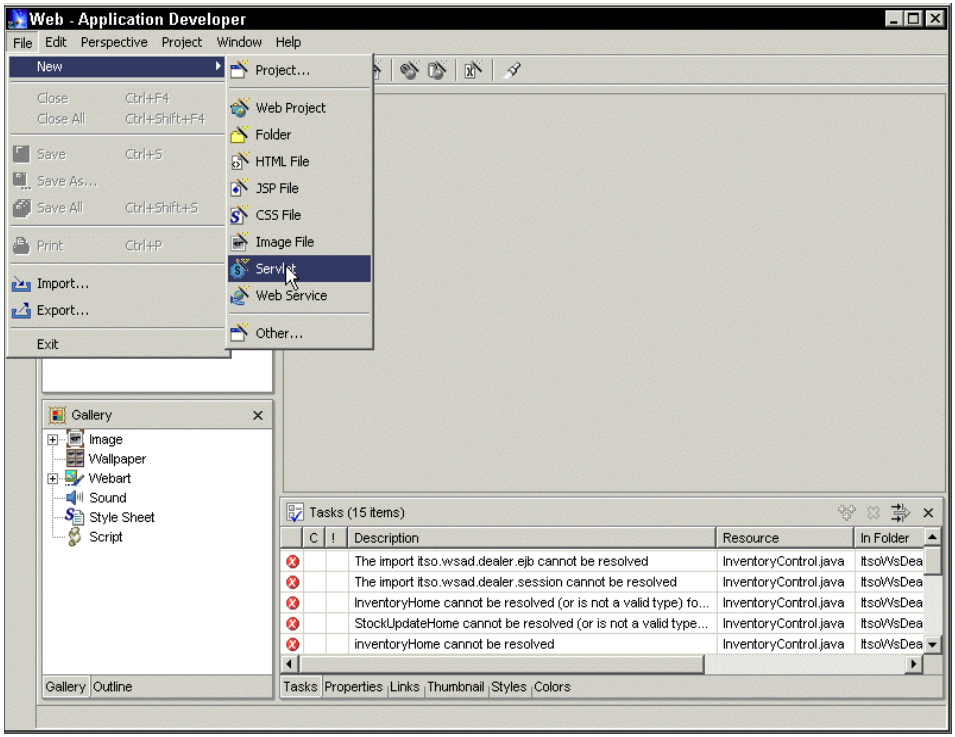


Figure 8-2 Adding new servlet from the Web perspective

8.2.3 Creating a simple servlet

You will be presented with the Create the Servlet Class dialog, where you need to supply the following information (Figure 8-3):

1. The **name** of the servlet (use SimpleServlet for our example) and the **folder**, where it should be created.
2. The **Web project** where the servlet will be placed, (your current project should be preselected), the package that the servlet will belong to, (the servlet is added into a default package if you do not specify another one), a name for the servlet, and its superclass. A servlet created through this wizard can have HttpServlet, or any class that has HttpServlet in its super-hierarchy, as its superclass.HttpServlet is the pre-selected default in the dialog.
3. A **modifier** to specify whether your servlet class is public, abstract, or final. (Classes cannot be both abstract and final.)
4. Whether the servlet you create implements the Single Thread Model **interface** by selecting the **Use Single Thread Model** option. This guarantees

that there will not be simultaneous access to the same servlet instance, which tends to stabilize thread execution ordering.

5. Any **additional interfaces** that should be implemented.

   Click the **Add** push button to open the **Superinterfaces Selection** dialog. Start typing the name of the interface that you want to add in the **Choose interfaces** field, and the list of available interfaces listed in the **Matching types** list box will be updated dynamically to display only the interfaces that match the pattern.

6. **Select** any appropriate **method stubs** to be created in the servlet code.

   The **Inherited abstract methods** and **Constructors from superclass** options add stubs for abstract methods inherited from the superclass and superclass constructors that must be implemented, (unless you intend to create an abstract servlet).

   Select to create the init() method. The doGet() and doPost() methods should be already preselected.

Press **Next**.

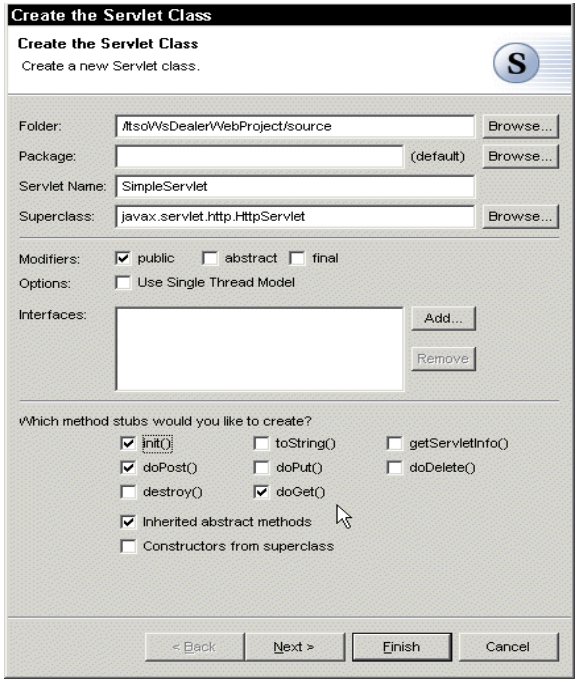


Figure 8-3 Create the Servlet Class dialog - page 1

The next page - **Define the Servlet in the Deployment Descriptor (web.xml) file** page, will be displayed (Figure 8-4). Here you have the opportunity to add servlet deployment information to the Web project's web.xml file.

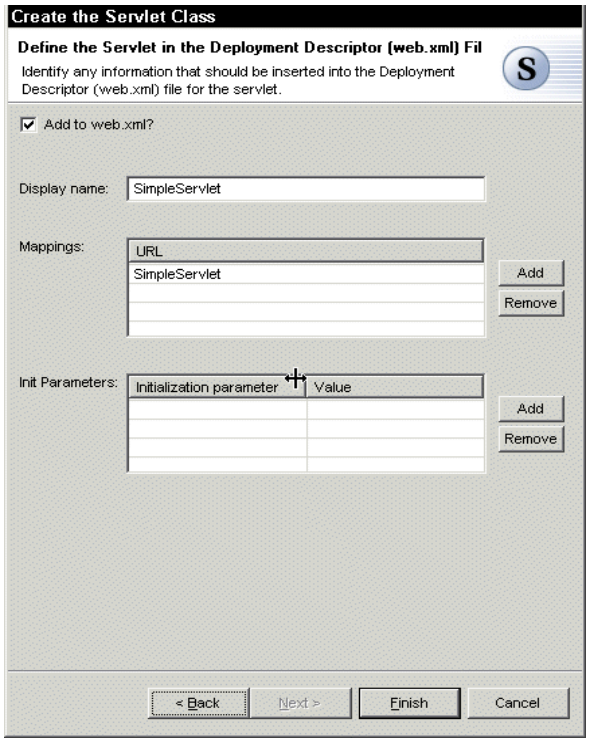


Figure 8-4 Create the Servlet Class dialog - page 2

If you select the **Add to web.xml** check box, the servlet, along with its display name and any URL mappings and initialization parameters associated with the servlet, will be automatically included in the project's web.xml file.

The **Servlet Name** value provided in the previous page of the wizard is automatically displayed on this page. The name will be updated when you change the value in the **Servlet Name** field.

After defining your mapping you can click **Finish**.

The servlet is generated and added to the project. In the Navigator view you can see the servlet displayed at its position in the file hierarchy. An editor is opened in the Top Right pane, where you can view and edit the generated servlet source code (Figure 8-5).

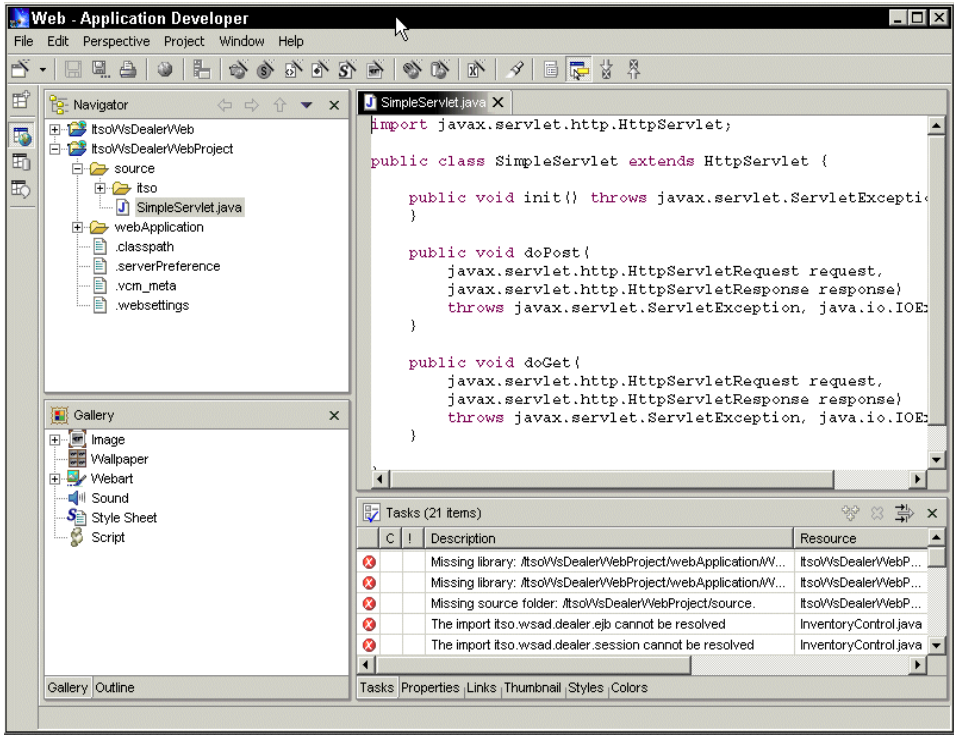


Figure 8-5 Newly created servlet

8.2.4 Editing the servlet

Application Developer has no generated a skeleton servlet for you. Your task is now to add code to the servlet methods in order to implement the required behavior for your needs.

Note: The code for SimpleServlet.java can be found in the sample code for this Redbook, (`..\Chapter8\SimpleServlet.java`). If you want, you can now cut and paste the completed servlet code into the skeleton servlet that you have just created. In the following discussion we will look at the code and describe the methods that need to be written.

(If you instead choose to *import* the file containing the SimpleServlet code, you must then manually add the servlet entry to the web.xml file.)

In the previous chapters we have shown you how to access the Parts database to display its content using a Java application and an applet. Now you will be achieving the same result using the servlet you have just generated.

To do this you will need to do the following:

- Replace the import statement at the beginning of your servlet class source code with a list of imports as shown in Figure 8-6.
- Complete the empty body of `doPost()` with a call to `doGet()`. The code of the methods should now look as shown at Figure 8-6.

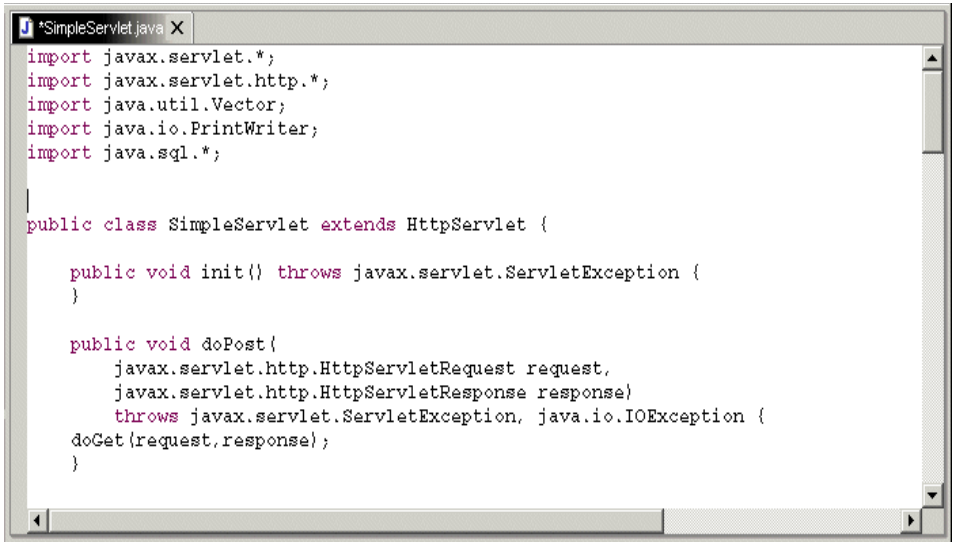


Figure 8-6 SimpleServlet imports and doPost() method code

- Add a new method, getResults(), that accesses the database and selects the parts according to a provided search criterion. The source code of this method is shown in Figure 8-7.

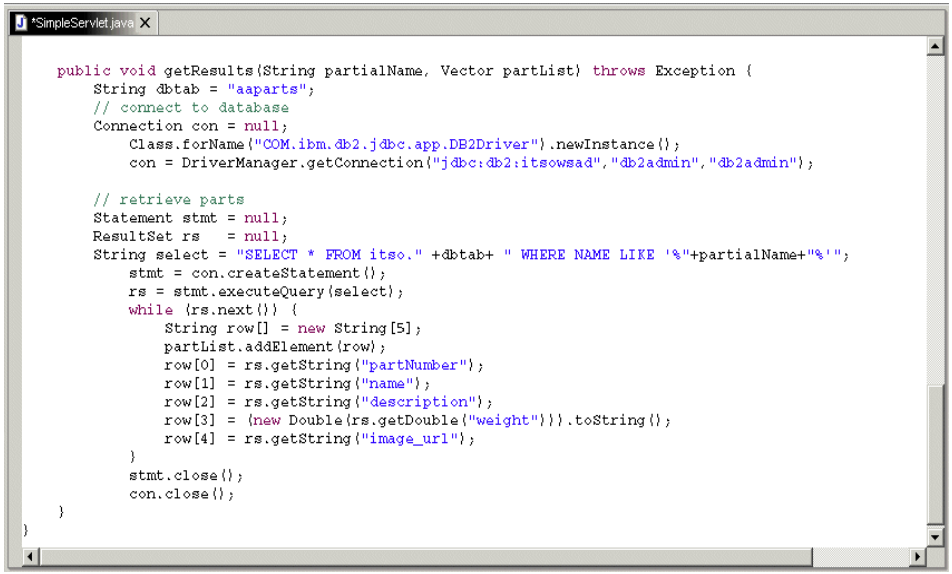
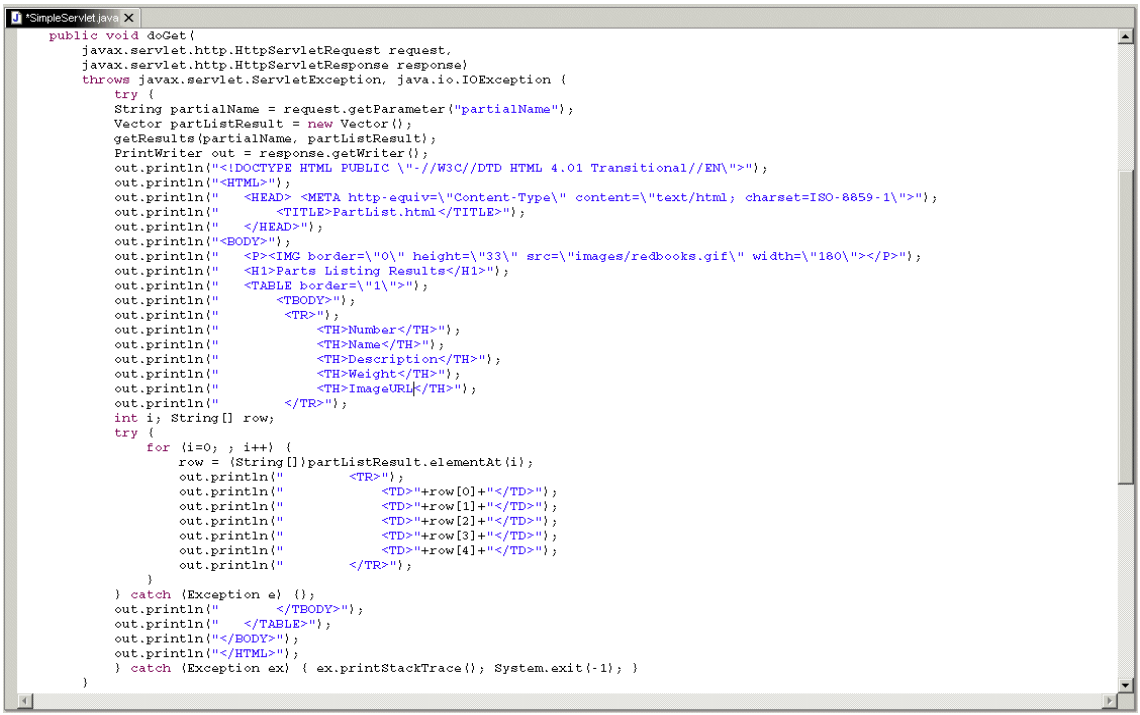


Figure 8-7 SimpleServlet>>getResults() method code

Finally you need to complete the empty body of the doGet() method with the code that performs the input processing, calls the database to retrieve the data, and generates the output HTML as shown in Figure 8-8.



```
*SimpleServlet.java
    public void doGet(
        javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
        throws javax.servlet.ServletException, java.io.IOException {
            try {
            String partialName = request.getParameter("partialName");
            Vector partListResult = new Vector();
            getResults(partialName, partListResult);
            PrintWriter out = response.getWriter();
            out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");
            out.println("<HTML>");
            out.println("   <HEAD> <META http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">");
            out.println("       <TITLE>PartList.html</TITLE>");
            out.println("   </HEAD>");
            out.println("<BODY>");
            out.println("   <P><IMG border=\"0\" height=\"33\" src=\"images/redbooks.gif\" width=\"180\"></P>");
            out.println("   <H1>Parts Listing Results</H1>");
            out.println("   <TABLE border=\"1\">");
            out.println("       <TBODY>");
            out.println("        <TR>");
            out.println("            <TH>Number</TH>");
            out.println("            <TH>Name</TH>");
            out.println("            <TH>Description</TH>");
            out.println("            <TH>Weight</TH>");
            out.println("            <TH>ImageURL</TH>");
            out.println("        </TR>");
            int i; String[] row;
            try {
                for (i=0; ; i++) {
                    row = (String[])partListResult.elementAt(i);
                    out.println("        <TR>");
                    out.println("            <TD>"+row[0]+"</TD>");
                    out.println("            <TD>"+row[1]+"</TD>");
                    out.println("            <TD>"+row[2]+"</TD>");
                    out.println("            <TD>"+row[3]+"</TD>");
                    out.println("            <TD>"+row[4]+"</TD>");
                    out.println("        </TR>");
                }
            } catch (Exception e) {};
            out.println("       </TBODY>");
            out.println("   </TABLE>");
            out.println("</BODY>");
            out.println("</HTML>");
            } catch (Exception ex) { ex.printStackTrace(); System.exit(-1); }
        }
```

Figure 8-8 SimpleServlet>>doGet() method code

8.2.5 Testing the servlet

To test the SimpleServlet you have to do the following:

Change the action name in the PartList.html file from **PartList** used in previous chapter to **SimpleServlet** (Figure 8-9).

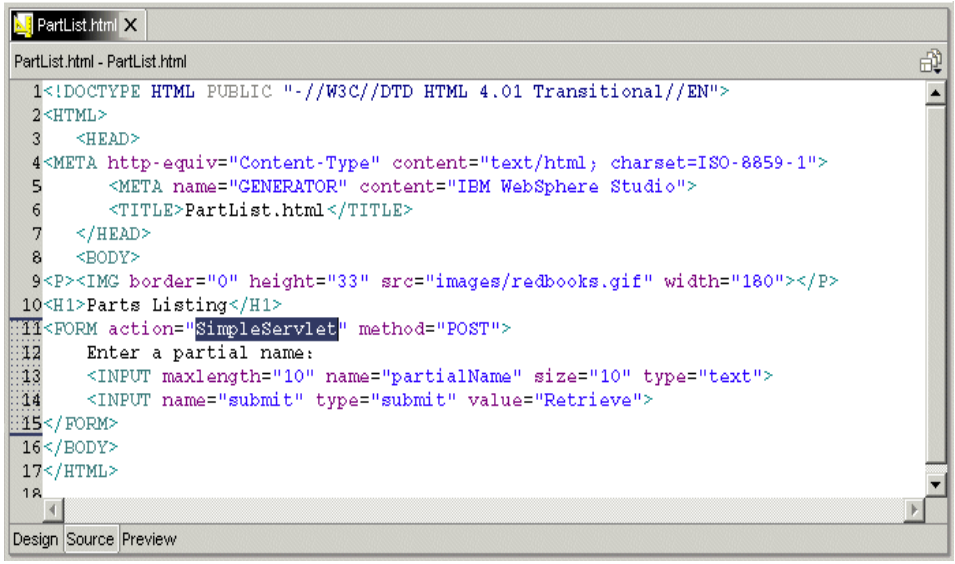


Figure 8-9 Changing the action name in PartList.html

Launch the PartList.html file in the local test environment. You do this by selecting **Run on Server** from the context menu of Partlist.html in the Navigator view of the Web perspective (Figure 8-10).

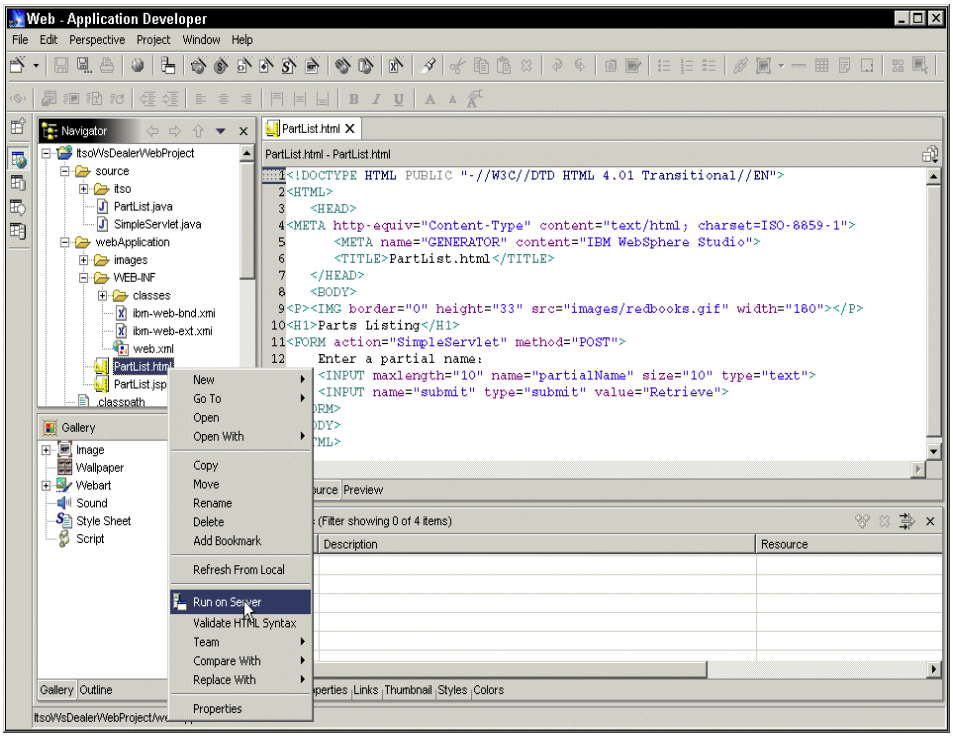


Figure 8-10 Starting the SimpleServlet unit test

If required the server will be synchronized and started at this point. (If this is the case, Application Developer will switch to the Server perspective.) Once this processing is completed, a Web browser window will be opened in the top right pane of your workbench. It will display the PartList.html page, which allows you to enter the Part search criteria. Enter ‘RR’ and click **Retrieve** (Figure 8-11).

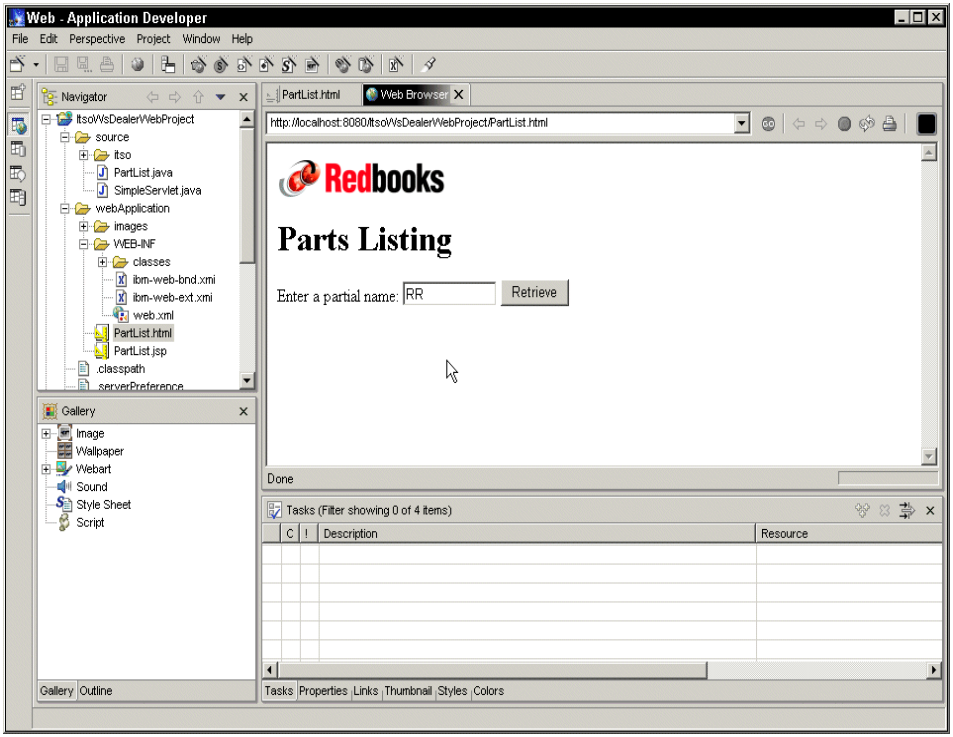


Figure 8-11 Starting from the PartList.html

If there are no problems with the servlet code, the query results will be displayed in the browser window (Figure 8-12).

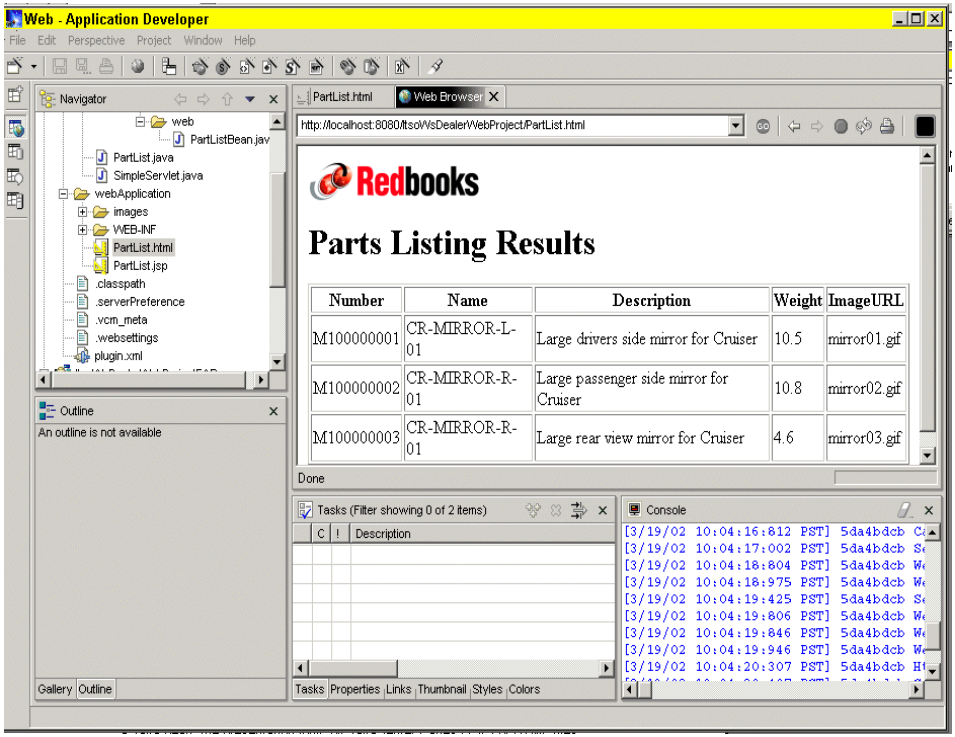


Figure 8-12 SimpleServlet test result

8.2.6 Conclusion

As you have seen from the preceding example, a servlet can be a self-contained program. However, you can choose to instead split the application into two parts:

1. **Business logic** (content generation), which governs the relationship between input, processing, and output.
2. **Presentation logic** (content presentation, or graphic design rules), which determines how this information will be presented to the user.

Using this model, you may choose to have your application business logic handled by a JavaBean and the presentation logic by one or more JavaServer Pages, (JSPs), or HTML files. The servlet would now only handle the HTTP protocol.

Note: JSPs files, too, could be used to manage both the presentation and business logic for a Web application. JSP files use structured markup for presentation, and supply servlet model behavior at runtime.

Over time a standard approach has evolved in constructing non-trivial Web Applications - the usage of the *Model-View-Controller (MVC)* pattern. We will now look at this pattern and how it relates to Web development.

8.3 Model-View-Controller pattern

The Model-View-Controller pattern, Figure 8-13, was developed in the mid-1980's by developers of the Smalltalk GUI library and is a model for how to display the same data in several windows in different views. The user can manipulate the data in any window and the other windows will immediately reflect any changes.

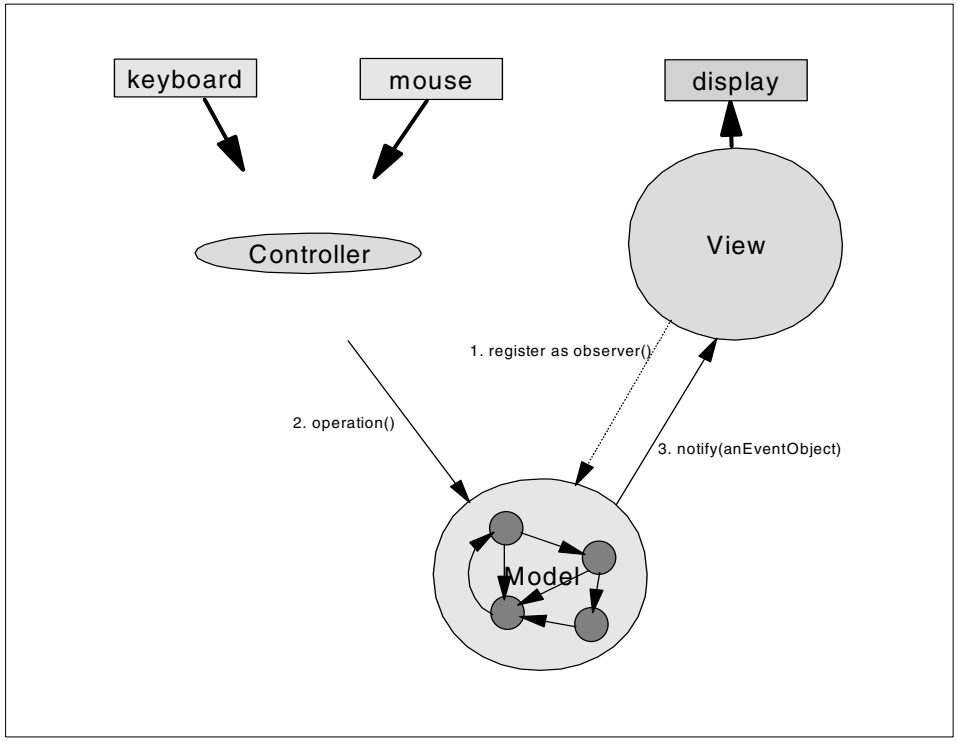


Figure 8-13 MVC pattern

The three components of MVC are described in the following section.

8.3.1 Model

When you hear the term 'model', you should think "data model". The Model encapsulates application state information (data) and operations that modify that state.

A model is meant to serve as a computational approximation or abstraction of some real world entity, process, or system. For example, the Model of a shopping cart in an e-commerce application contains the identity and number of purchased items and offers operations to add and remove items from the shopping cart.

Code that interacts with business data (for example databases) should be part of the model. Typically the model component will be written as a Java JDBC application and will be referred to as a Java bean, DataBean, or simply a Bean.

8.3.2 View

The view is that part of your code which produces the visual components (for example GUIs), through which the user interacts with your application.

The View is responsible for presenting (rendering) the information contained in the Model in a particular output modality. For example, a graphical view may render a shopping cart as a list showing the items and their number, plus a "remove" button next to each item.

A View is an Observer of the Model, and therefore also depends on the Model. At initialization time the View registers with the Model to express its interest in any changes of the Model's state. Whenever the Model changes state it notifies the View by sending it an EventObject that contains the state change. The View then updates its rendering.

When the view is rendered by a Web browser, the language passed to the browser must be HTML or XML. Often other technologies are used on the front end to generate HTML more productively. Examples of such technologies which produce HTML are JavaScript, Servlets, and JavaServer Pages (JSPs).

8.3.3 Controller

As its name implies, the controller component controls the overall flow. The controller code interacts with the view and model components to deliver a modular yet integrated solution.

It is the Controller that accepts input from the user in a particular modality, interprets that input (the interpretation may depend on the View), and invokes the appropriate operation on the Model. For example, when the Controller detects a mouse click event on the "remove" button of an item it invokes the remove operation on that item. Any state changes that this operation causes on the Model are sent by the Model to the registered Views via events.

The controller component is normally written in Java and implemented as a Servlet.

8.3.4 MVC usage rules

In order to support reusability, the interactions which do occur should be well defined and the dependencies between the elements (M-V-C) should be minimized.

One of the goals of the MVC pattern is to enable the combination of a single Model with multiple Views and Controllers. The MVC pattern ensures that the Views are kept synchronized. When the Controller recognizes a valid command from the user's input, it invokes the corresponding method on the Model. The Model verifies that the operation is compatible with its current state, executes it and changes the state of the Views correspondingly. The views, as they have registered themselves as observers, get now informed about the Model’s state change and update their rendering correspondingly.

The dependencies must be kept minimal

To support multiple views and controllers the dependencies must be kept minimal.

Note: A is said to be dependent on B when the code of A embeds knowledge about B.

This leads to the following rules Figure 8-14:

1. The Model does not have any dependency on Views or Controllers.
2. A View depends on its associated Model. It has to know the structure of the Model's state to be able to render it.
3. A View does not have a dependency on Controllers. Therefore several different Controllers can be associated with the same View.
4. A Controller depends on its associated Model and View. The Model defines the operations the Controller can invoke and the View defines the context in which the Controller interprets the user input. This makes the Controller tightly coupled to the View.

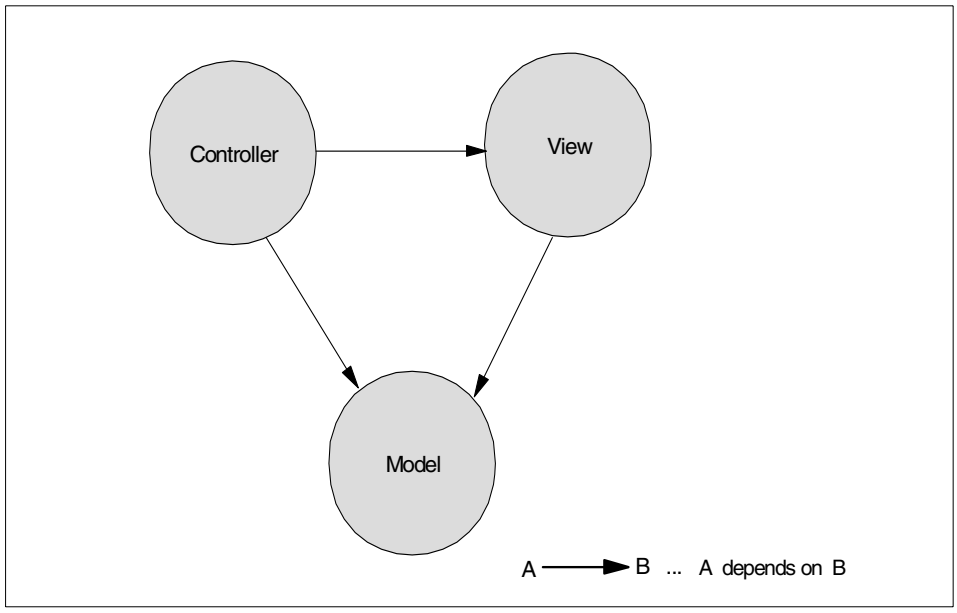


Figure 8-14 Dependencies allowed in the MVC pattern

The interactions must be kept minimal

Another precondition to support multiple Views and Controllers is to keep interactions minimal.

In particular a Controller must never directly affect the rendering of its associated View. Instead user input must make a complete round trip through the Model before its effects become visible in the View. This rule guarantees that a state change updates all Views and that the Views remain synchronized. Often implementations with a single Controller violate this rule because of sloppy thinking: "I already know that this state change will occur, and therefore do not need wait for the Model to tell me about it". This is wrong for two reasons:

1. The Model can veto the operation for some reason. The operation will not occur.
2. Other Controllers may concurrently invoke operations on the Model. Some other operation can slip in between, which fundamentally changes the rendering and makes any assumptions about it invalid.

In addition it is impossible to extend such shortcut implementations later with additional Controllers.

8.3.5 The MVC pattern in Web applications

Although the MVC pattern was originally devised for the organization of fat client GUI libraries, it has in the past several years received widespread acceptance as a suitable architectural pattern for implementing Web based solutions, too. Its structure has been applied, (with limitations), in recent Web applications. This is not surprising, since in both cases the separation of concerns is the driving force behind architectural choices.

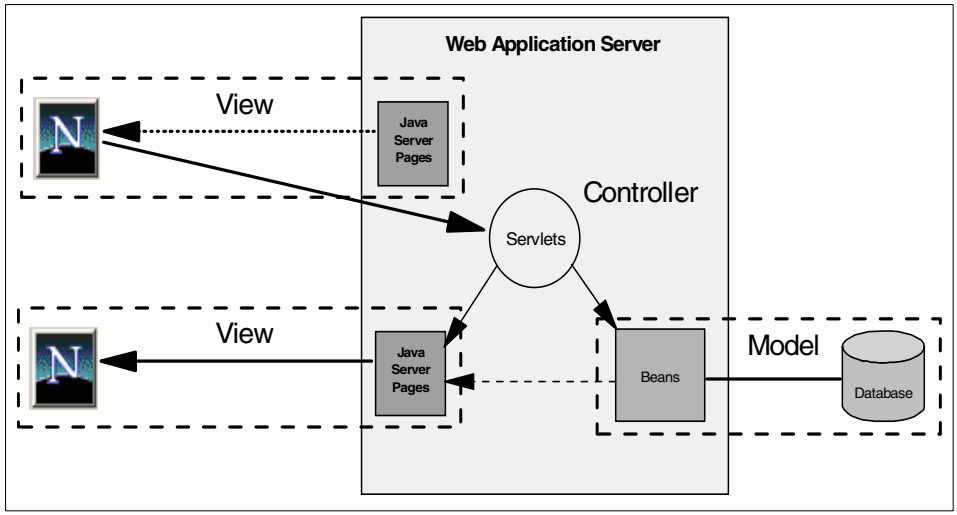


Figure 8-15 MVC in a three-tier configuration with a Web application server

8.3.6 Extending the MVC pattern to distributed applications

Although the MVC pattern was originally devised for GUIs running on a single machine, it can be extended relatively straightforward to distributed systems, where some interfaces between Model, View and Controller may cross the network. The placement of the Model, the View and the Controller then becomes a crucial issue.

The **client-centric** approach, Figure 8-16, puts all three functions: Model, View and Controller on each client device. The Model, which exists conceptually only in one instance, is “replicated” among the devices. A replication protocol keeps the copies of the Model synchronized.

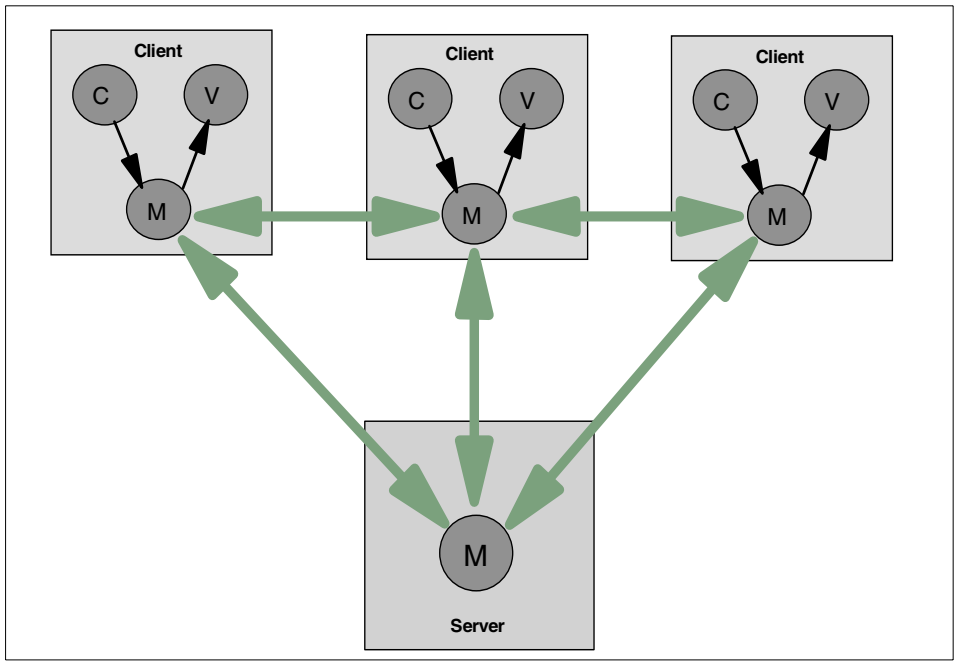


Figure 8-16 Client-centric approach

The **server-centric** approach, Figure 8-17, puts both Controller and Model on a server. The client devices contain only the Views.

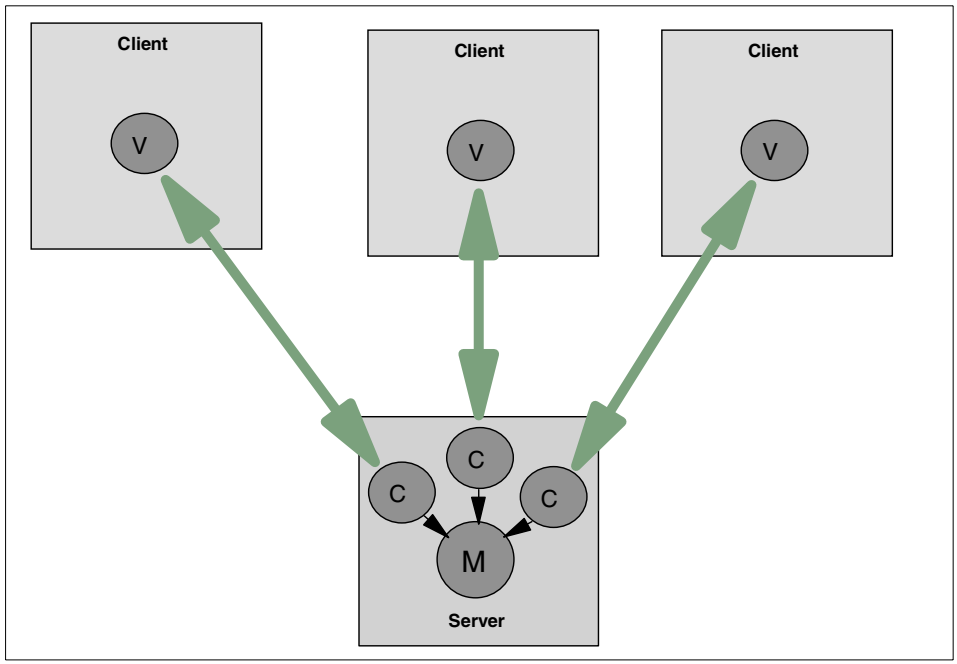


Figure 8-17 Server-centric approach

8.4 Developing Web applications using the MVC pattern

One of the great challenges Web application programmers face is designing and implementing solutions which unite numerous disparate elements. It is not uncommon for a single Web solution to combine static HTML pages, JSPs, JavaScript, Servlets, and JavaBeans.

MVC is basically a blueprint which programmers should use to design and implement the Web solutions. Using a consistent architecture allows developers to deliver well designed, modular, high quality solutions.

Another benefit of MVC is that HTML code can be clearly separated from Java code. This is very important, since the Web developer's job is normally clearly differentiated from that of an Application programmer. Web page modifications to HTML nested within a Java servlet can be challenging to a Web developer.

8.4.1 Creating the model Java bean

Based on the discussion in the previous section, you will restructure your auto parts listing Web application.

As shown in Figure 8-14 on page 221, there are dependencies defined and allowed between the elements of the MVC pattern. To start in the right place you should first create the one, that does not depend on any other two - that is the **Model**.

In our simple example the Model will be implemented as a Java bean. This object will have the responsibility for establishing the connection to the database and performing the table access, that is executed as an SQL select statement.

Note: The completed source code for the Java bean class be can found in:`..\Chapter8\itso\wsad\dealer\web\PartListBean.java`. You may already have the Java bean loaded in your workspace from 7.4, “Creating Web pages from a JavaBean” on page 182. If you want to follow through this example and create the Java bean, remove PartListBean.java from your workspace.

To define the Java bean you need to add a new class to your Web application.To do so, first select the source subdirectory below the ItsoWsDealerWebProject project and click on the **Open The New Wizard** icon  . Select **Java** and **Java Class** in the New dialog as shown inFigure 8-18.

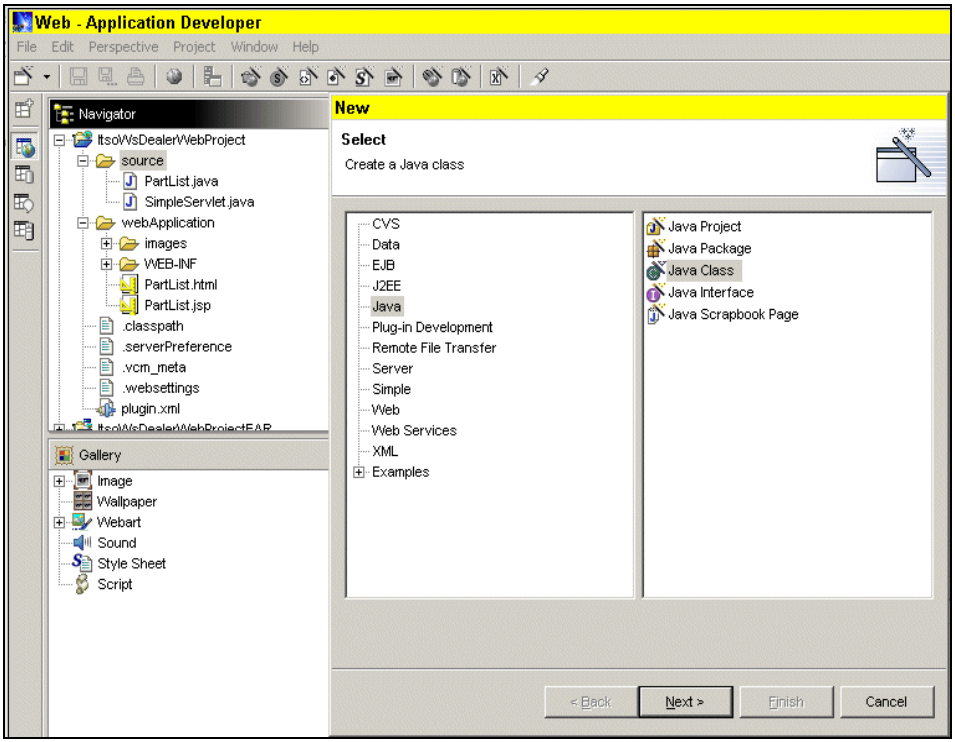


Figure 8-18 Starting a new wizard while adding the PartListBean class

Press **Next**.

A Class Definition dialog will be opened for you. Enter `PartListBean` as the name of the new class. Notice, that the folder will be correctly selected, (it is the one that was selected before clicking the **Open The New Wizard** icon  . If you had not selected the right folder prior to starting the wizard, you could do it now by entering the right path, or clicking the **Browse** button to navigate to and select the folder.

For the package enter `itso.wsad.dealer.web` and select the **Constructors from superclass** and **Inherited abstract methods** checkboxes. The **Finish** button should now be enabled (Figure 8-19). Click **Finish**.

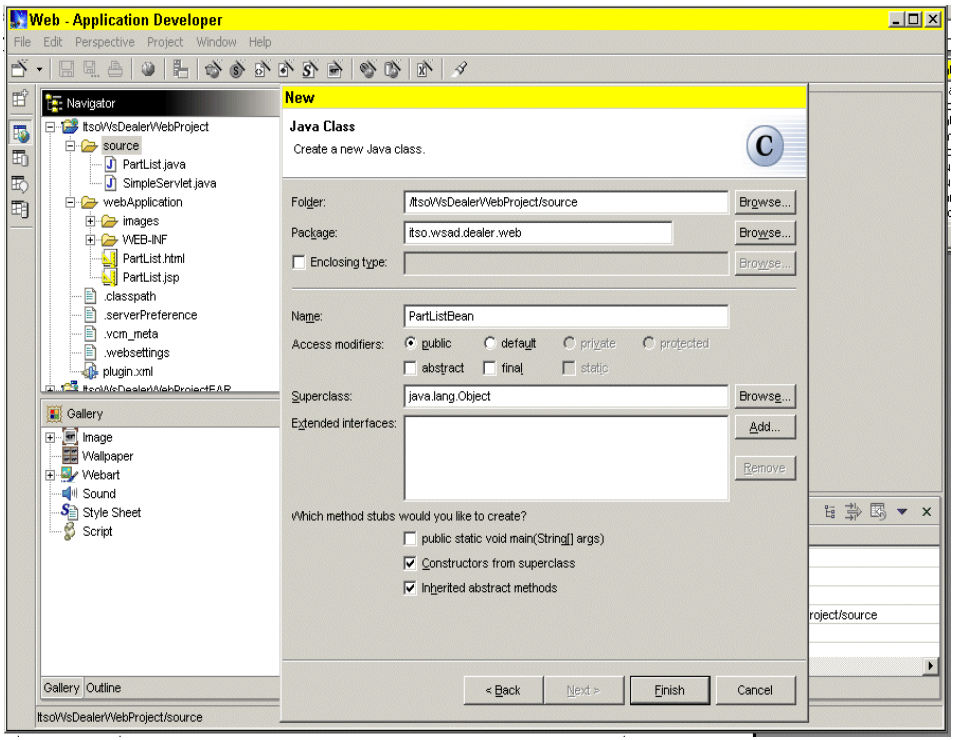


Figure 8-19 Selecting the folder and entering the package

The class will be created and included into the project. It is displayed in the Navigator view under the source folder and an editor will be opened to allow you to start editing it. As you can see the source contains the package definition and a constructor created automatically. As the superclass of this class does not define any abstract methods, no other method stubs were generated.

First of all you have to add the import statements again and then define a variable called partList of class Java.util.Vector. See the code in Figure 8-20.

PartListBean.java

```
package itso.wsad.dealer.web;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Vector;

public class PartListBean {


    java.util.Vector partList = new java.util.Vector();

    /**
     * Constructor for PartListBean
     */
    public PartListBean() {
        super();
    }
```

Figure 8-20 PartListBean source code - imports and variable definition

As you can see the constructor method remains unchanged. It simply calls its superclass constructor.

Having declared the variable you can now show to the Outline view. You can change to the Java perspective or if you want to add the Outline view to the current perspective, you can activate it by selecting **Perspective—>Show View—>Other—>Basic—>Outline**. In the Outline view you can see the current structure of the class. You can select the partList variable from here, and use its context menu to have Application Developer generate the getter and setter method for it (Figure 8-21).

The result of the method generation is shown in Figure 8-22. Note that you can see the generated methods both in the source code editor and in the Outline view.

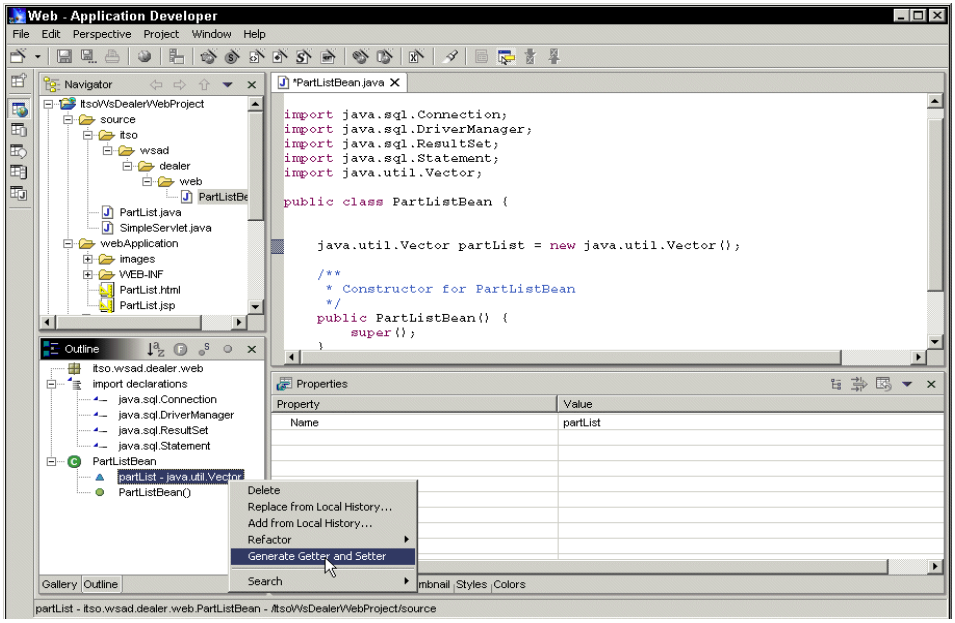


Figure 8-21 Generating getter and setter methods for a variable

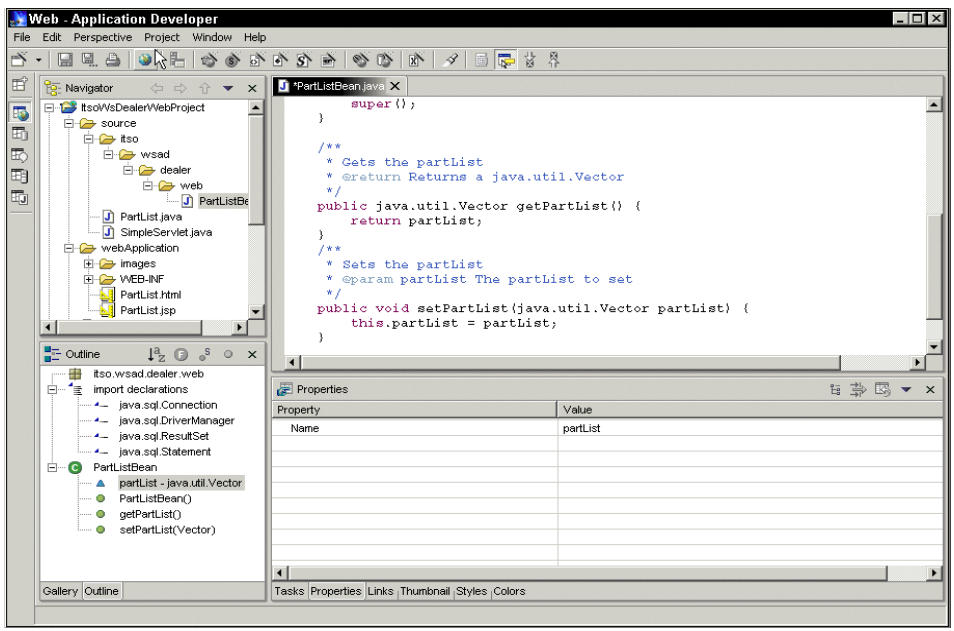


Figure 8-22 The getter and setter methods generated

Next you can enter the code for the business logic. In this case it is fairly simple. You need to connect to the database, execute a select statement, and assign the result of it to the partList variable.

First you will need to create the `connect()` method for connecting to DB2. You will connect to the database named `jdbc:db2:itsowsad` using the `COM.ibm.db2.jdbc.app.DB2Driver` using a valid user ID and password. The code to do this is shown in Figure 8-23.

Note: You must enter a valid user ID and password that can connect to DB2 on your system. Our example shows the user ID and password as blank.

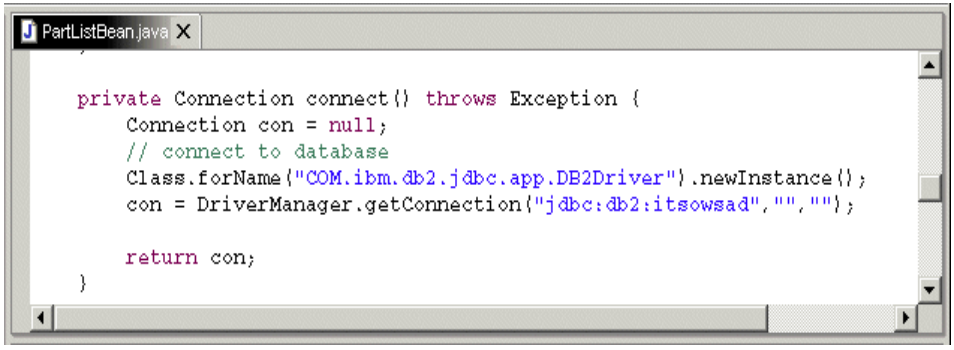


Figure 8-23 PartListBean connect() method

Last you need to create the select() method to execute the DB2 select statement and retrieve the results. For this you need to get the database name, the selection criteria from the parameter passed in to this method, along with a result variable where you will store the query result as an array of strings. The source code for this logic is shown in Figure 8-24.

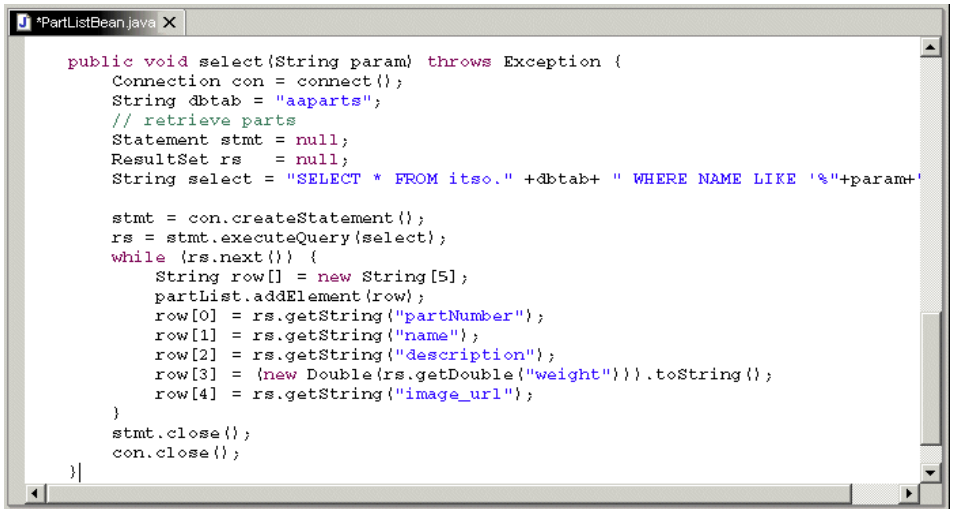


Figure 8-24 PartListBean select() method

With the Model class defined, you can now switch your attention to the View class. Both the Model and View classes will be referenced in the code of the Controller, so we choose to implement them first. This avoids dealing with unresolved reference errors generated by the Java builder. However, there is of course nothing preventing us from developing the components in any order or in parallel.

8.4.2 Creating the view JSP

A JSP file can be created and edited in the same editor you used when building our HTML pages. When working with a JSP, Page Designer has additional elements that can be used. As well as text and images, you can add JSP tags, Java bean references and scriptlets containing Java code. To create a new JSP file, do the following:

To launch the New JSP File wizard, select **File—>New—>JSP File**.

Note: The completed JSP can be can found in:`..\Chapter8\PartList.jsp`

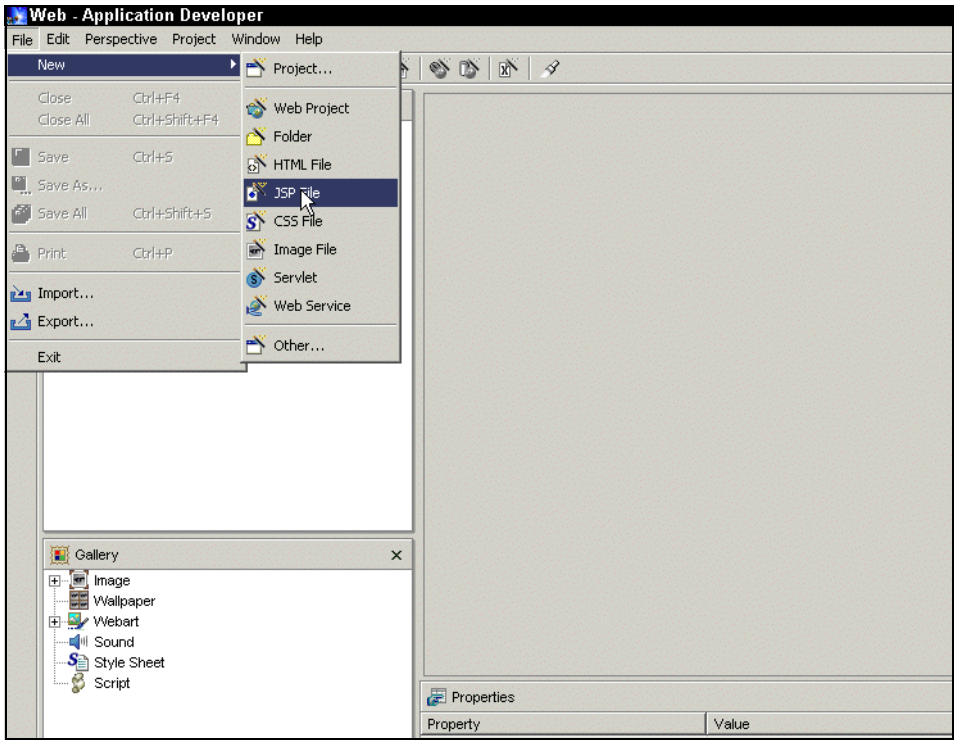


Figure 8-25 Adding a JSP to a Web project

As an alternative you can click on the **Create a JSP File** icon  in the workbench's icon bar.

You will be prompted to select an appropriate container for the file from the list of project folders and subfolders.

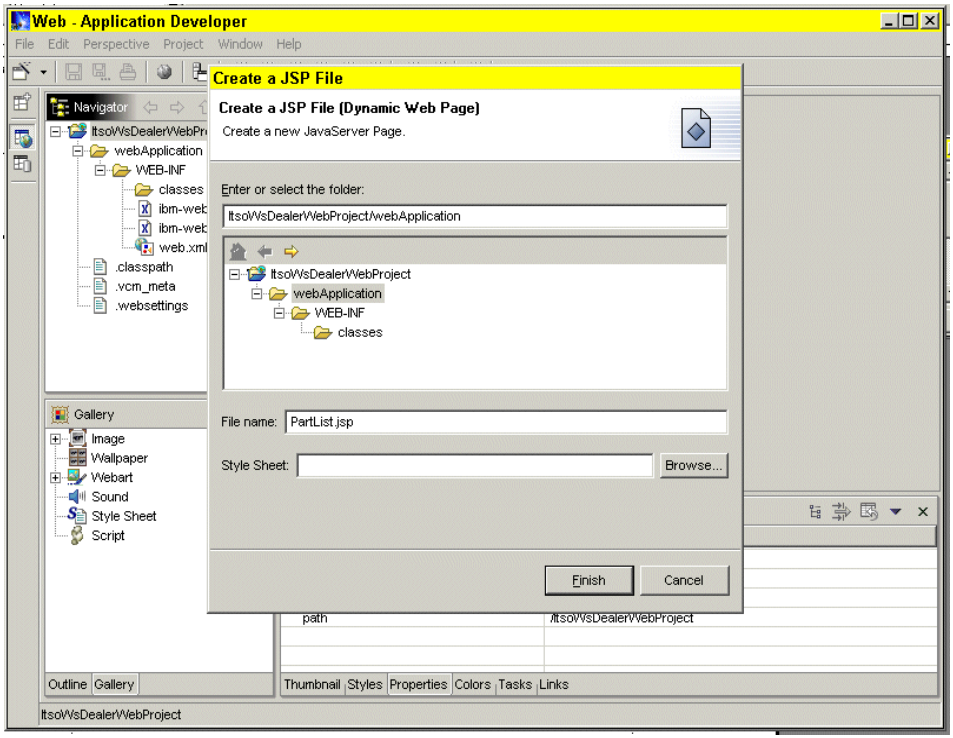


Figure 8-26 Creating a JSP file

The folder that you choose should be under the webApplication folder of your Web project.

Note: If an HTML or JSP file is *not* under this folder, then it *will not be included* in the WAR file that will be deployed to the server. In addition, link validation will not include files that are not under the webApplication folder.

Type the file name, (`PartList.jsp`), into the appropriate field. Optionally you can use the Browse button to locate and specify a style sheet to be used when generating the JSP.

Click **Finish** to create the new JSP in the project that you have selected.

The file will be added into the project structure in the Navigator view, (under the folder you had specified), and it will be opened for editing in an HTML editor. The default is Page Designer and its Design view.

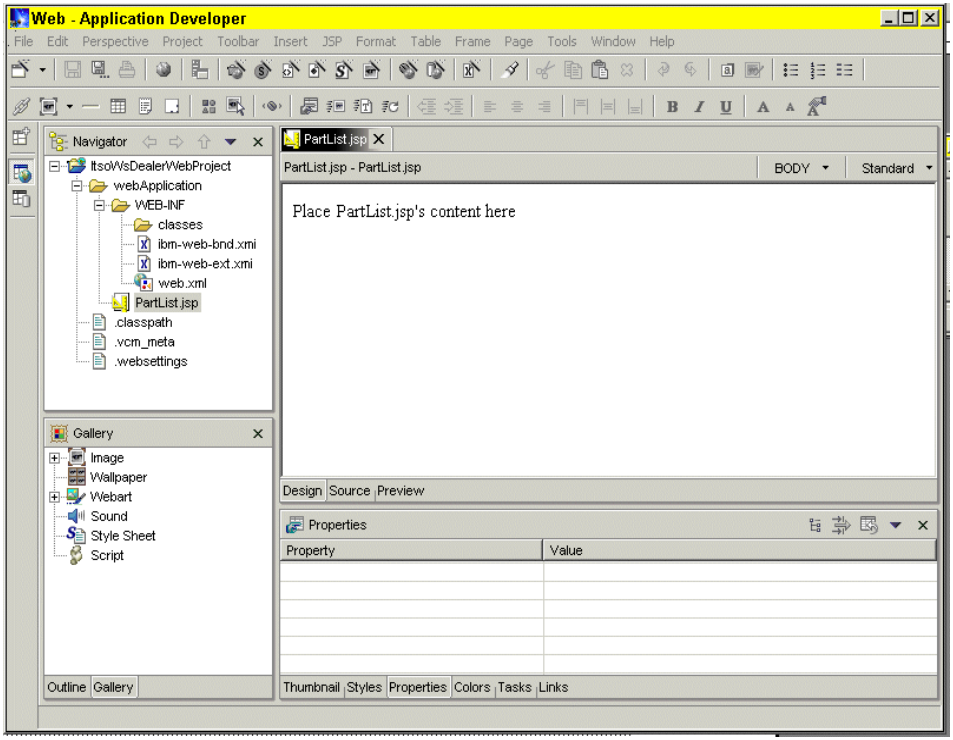


Figure 8-27 Newly created JSP opened for editing

As you can see the Page Designer has many menu options and icons available to support designing your new page as they were listed in detail in “Page Designer” on page 54.

Editing JSP file

In Application Developer you can create JSPs from scratch or use the Application Developer wizards to generate the skeletons and then edit them (you can add data from other JavaBeans, and customize the result tables, etc.).

The JSP files that can be created by the Application Developer wizards allow you to do the following:

- Access a Java bean when the page is processed.
- Embed variables in the page and get the value of bean properties at runtime.
- Format the variable data.
- Repeat a block of HTML tagging that contains embedded variables and HTML formatting tags.

You can customize the generated JSP files by adding your own text and images using JavaScript, HTML, or JSP scriplet tagging. These tags and script will be included in the HTML file created by the Web Server and returned to the requesting browser.

When a Web server such as WebSphere Application Server processes the JSP file, it performs the following actions:

- Preprocesses the JSP file into executable code.
- Instantiates the JavaBeans.
- Places the resulting data into the output page (replacing the special tags).
- Sends the results as an HTML stream to a Web browser or other output device.

When you want to edit an existing JSP in Application Developer, to open that file in Page Designer you can double-click a JSP file, or an HTML file having the JSP content embedded in it. Using the Page Designer **JSP** menu, you can insert the following JSP elements:

- Beans
- Expressions, scriplets, and declarations
- Forward and Include parameters
- Get and Set properties
- Plug-ins
- Custom tags

Let us now return to the JSP page that you have just created. When you are finished, the JSP should look as in Figure 8-28.

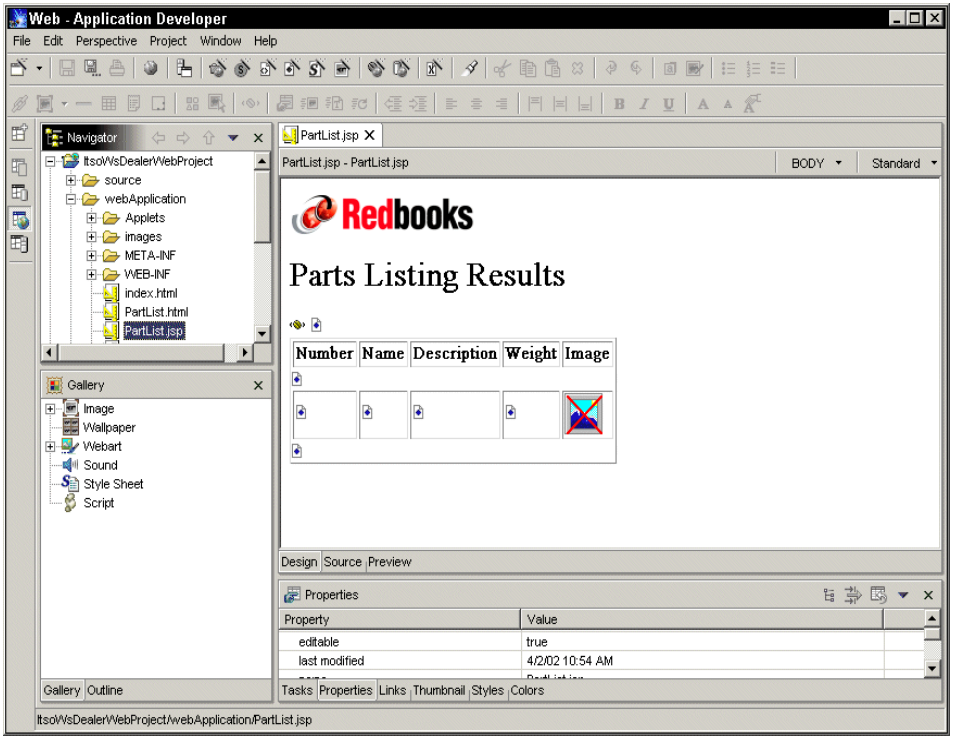


Figure 8-28 Final Parts.jsp look

The first JSP element that you will add to the page is the Redbooks image. You have learned in “Creating an HTML page with a form” on page 171 how to do this, so follow those steps to insert the image into your JSP.

Note: If you haven’t yet imported the sample image files, you should do so now. Create a sub folder named `images` under your webApplication folder, and import all the .gif files in the ..\Chapter7\images subdirectory.

Next you should insert a heading on the page. Position the cursor and type in the text (`Parts Listing Results`). Then select the text and bring up the context menu, select **Attributes** and change the style to *Heading 1*.

Now you want to insert a Java bean reference, (the jsp:useBean tag). To achieve that you can select **JSP —> Insert Bean** from the menu, or you can click on the **Insert bean** icon  in the toolbar. Either way you should first position the cursor in the page to where you want the bean inserted.

The Attribute dialog for the bean will be displayed (Figure 8-29).

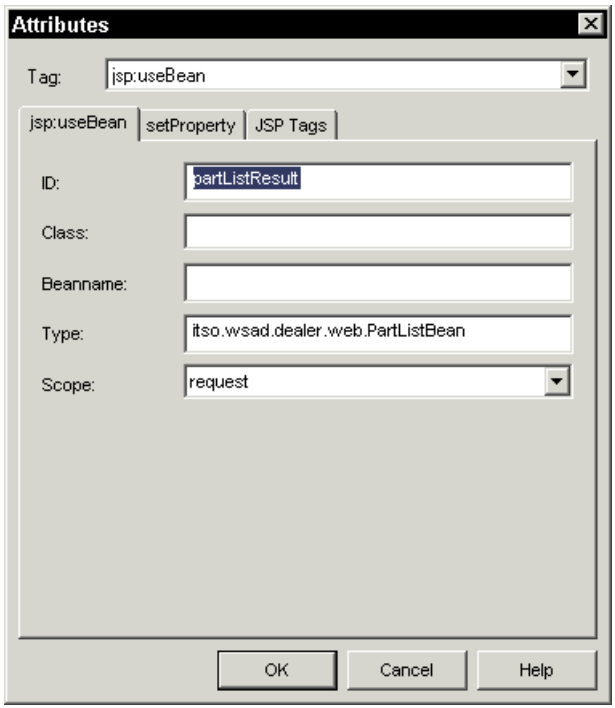


Figure 8-29 Inserting a JavaBean into a JSP

Enter `partListResult` as the **ID**, `java.util.Vector` as the **Type** and `request` as the **Scope**.

Figure 8-30 shows what your JSP should look like by now.

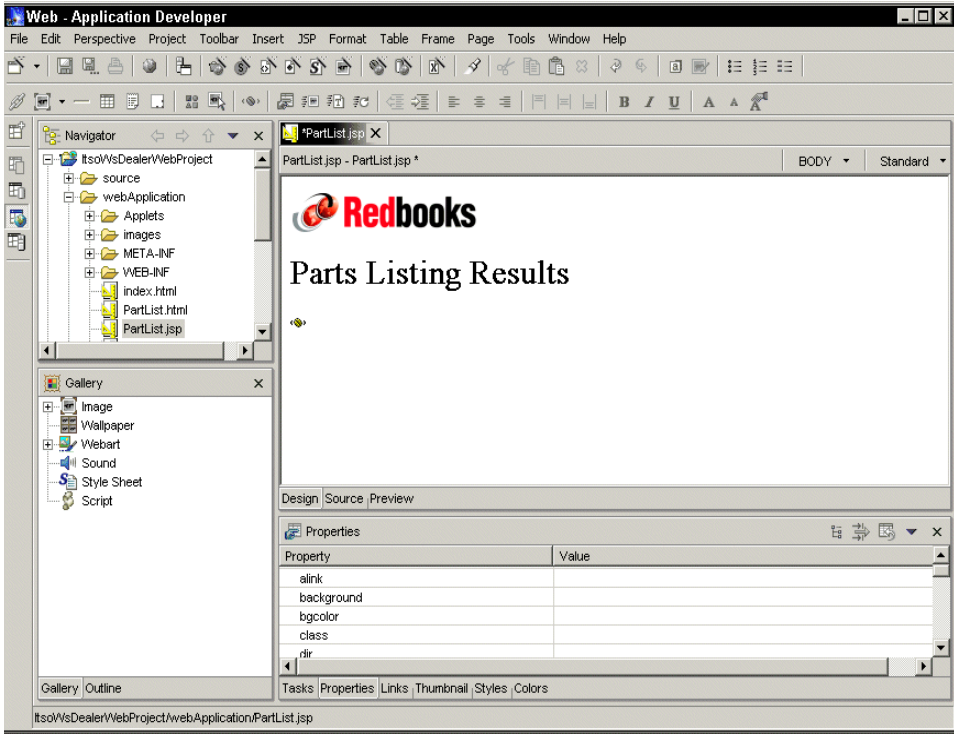


Figure 8-30 JSP under construction 1

Since this JSP will display a list of matches from the database, it will need to loop through and display the individual rows of the result set. To support this you need to define two variables now, one for the loop index and one to contain the results rows as an array of strings.

To do this you will insert a script into your JSP. Select **JSP —> Insert Scriplet** menu item from the Page Designer's menu bar.

A **Script** dialog will be opened for you (Figure 8-31). You first have to select the type of the script in the Language list box. Click on the arrow at its right end and select the **JSP Declaration** item.

Now you can enter the code in the left middle pane of the dialog. The code to be entered is:

```
int i; String[] row;
```

To complete the insertion of a script into your JSP click the **OK** button. The script dialog closes and an script icon will be added into your JSP.

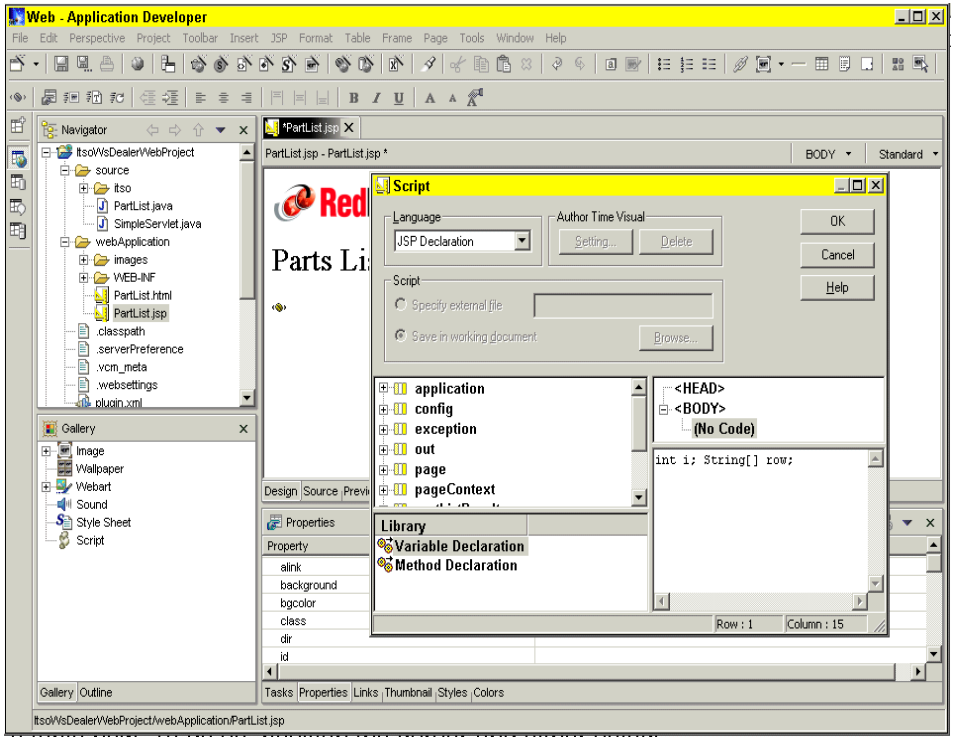


Figure 8-31 Adding a variables declaration to the JSP

Next you want to add the result list. It will be presented in the form of a table, so you need to insert a table into the page. To do so, position the cursor and either select **Table—>Table**, (or press **Ctrl+T**), or click the **Insert Table** icon  in the toolbar. An Insert Table dialog will be displayed. In this you can specify the number of rows and columns. Set number of rows to 2 and number of columns to 5.

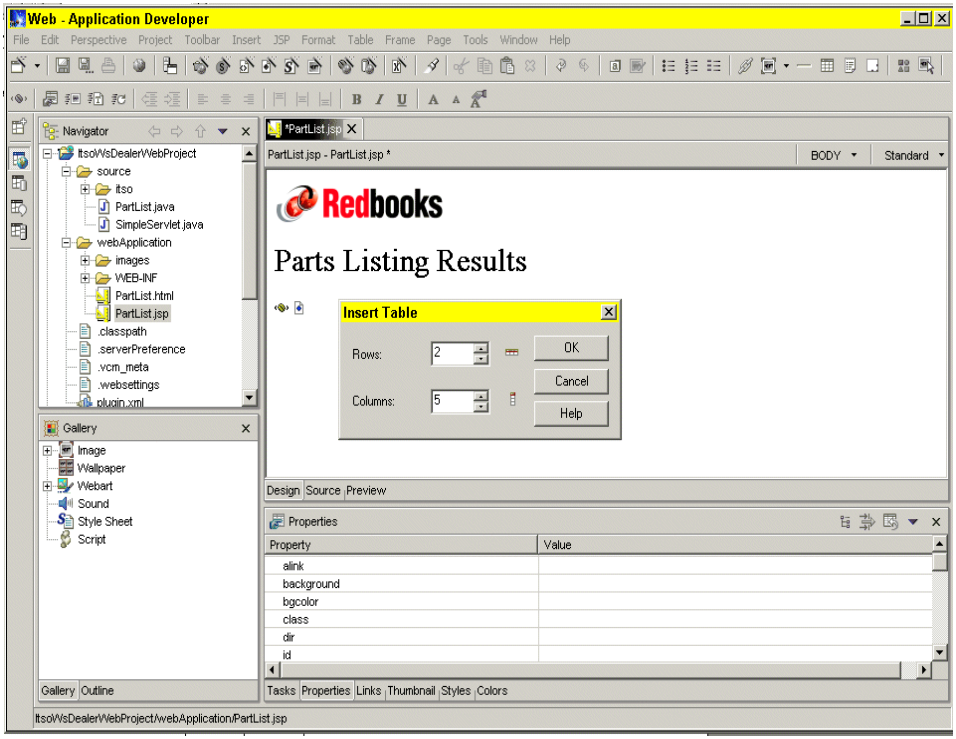


Figure 8-32 Inserting a table into the JSP

A table will be inserted and the cursor will be positioned in the first cell of the first row. You can start typing the headings of the individual columns. Type **`Number`**, **`Name`**, **`Description`**, **`Weight`** and **`ImageURL`** in the individual first row cells. You can switch to next cell using the **Tab** key each time. After you finished entering the labels, double-click on each of them and a property window for each cell will be opened. Here you can declare each of the first row cells as Header, which changes their layout.

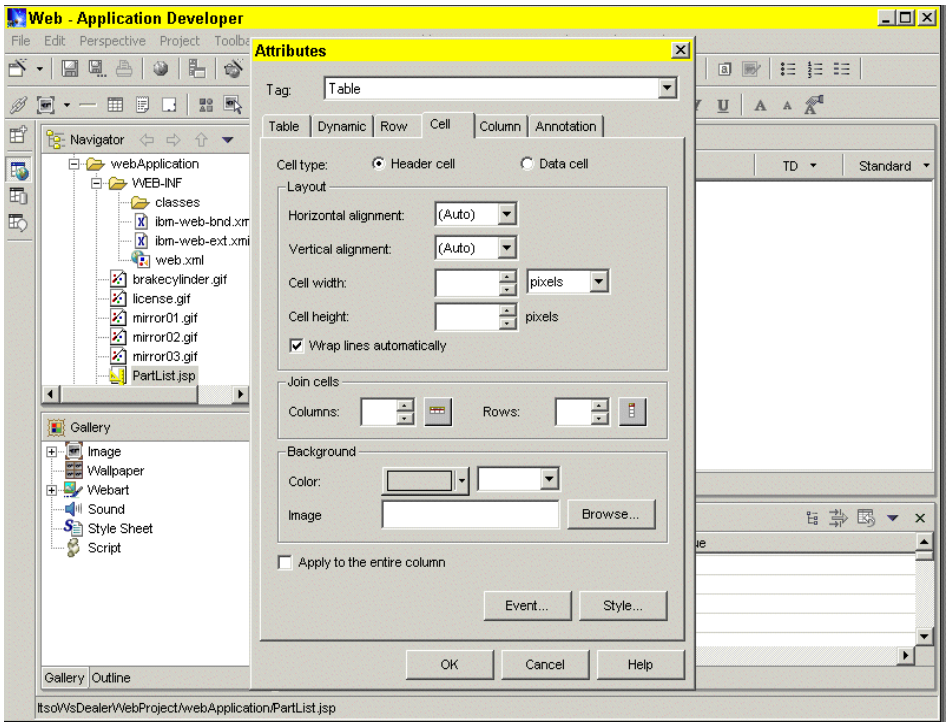


Figure 8-33 Declaring a table cell as header cell

After you have completed these steps the JSP should now look as in Figure 8-34.

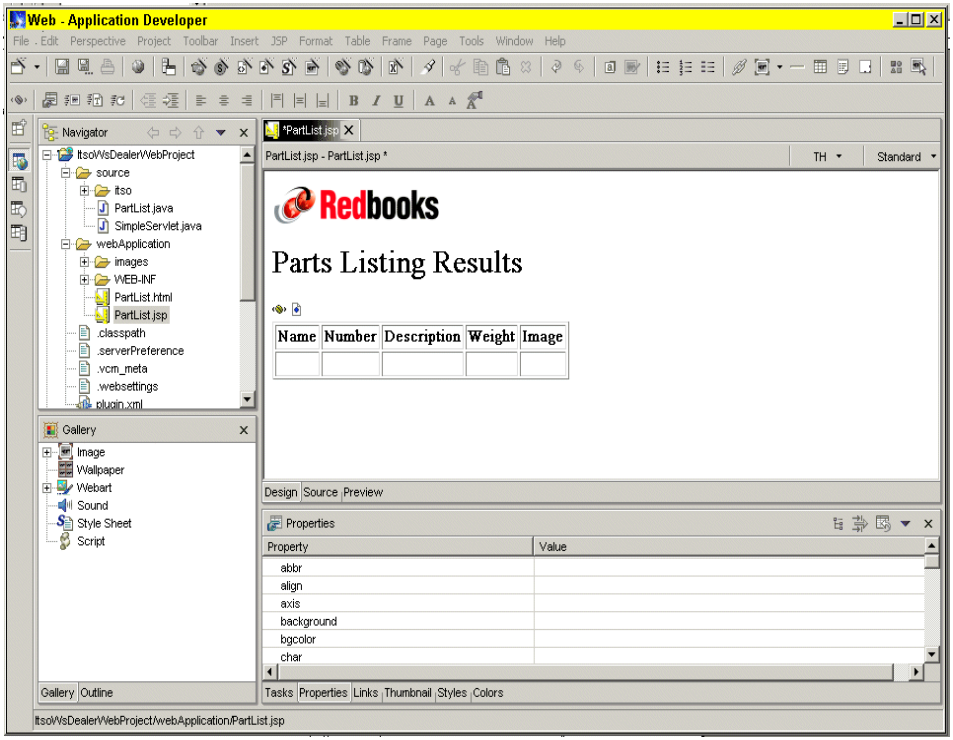


Figure 8-34 JSP under construction 2

Now you need to loop over the items in the partListResult bean and display the individual elements of it in multiple table lines. As there is no way to position the cursor at the right position in the Design view, you should switch to the Source view now and enter the loop code as a Java script in it.

To position yourself in the source code, as close as possible, you can set the cursor into the first cell in the second table row before you switch to the Source view. What you should see is shown in Figure 8-35.

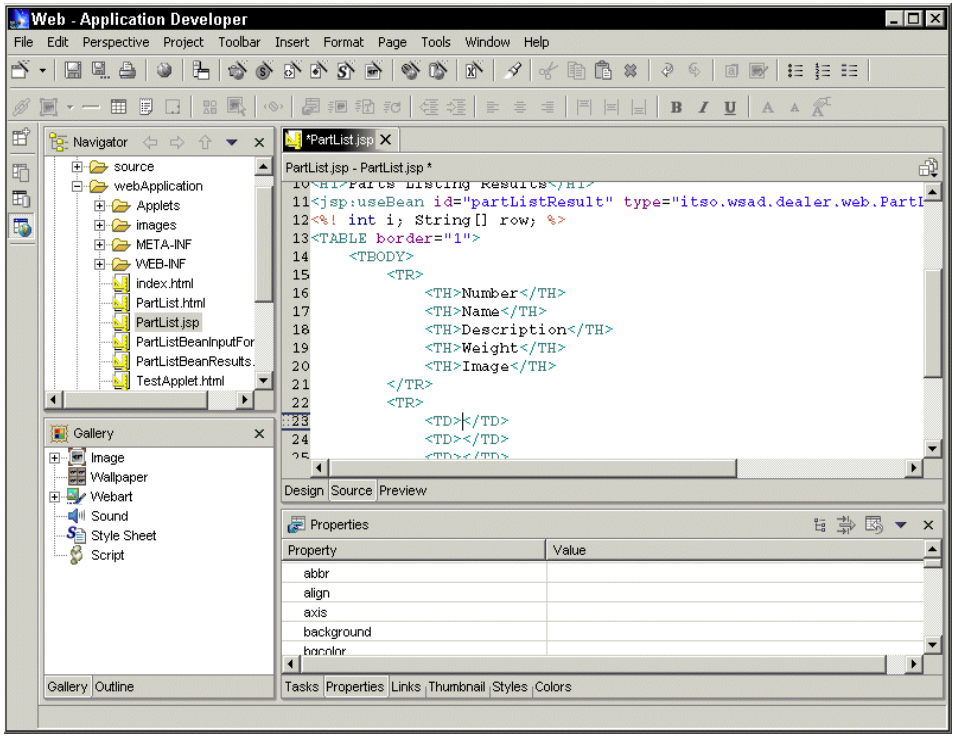


Figure 8-35 Switching to the source page

Notice the position of the cursor in the source page. It is positioned exactly between the <TD> and </TD> tags of the first cell of the second table row. So now you can easily position the cursor between the end of the first line and the beginning of the second (two lines above) and type in the script code for the loop. This is what you should type in:

```
<% try {
   for (i=0; ; i++) {
    row = (String[])partListResult.getPartList().elementAt(i); %>
```

Figure 8-36 shows how the code should look when this is done.

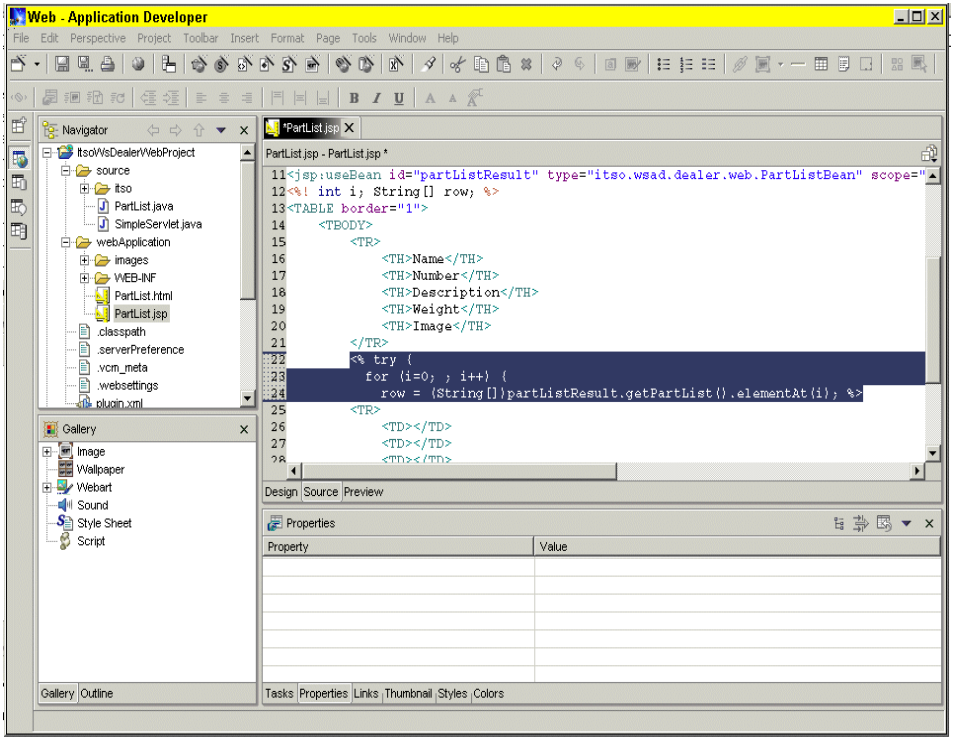


Figure 8-36 Entering script to JSP

Now switch back to the Design view and you'll find the script has been inserted between the header and content line of the table. Now you have to fill the cells of the content rows with the individual elements of the partListResult items (remember that you have defined the `row` variable to be an array of strings).

Position the cursor in the first cell of the second row again and select **JSP—>Insert scriplet**.

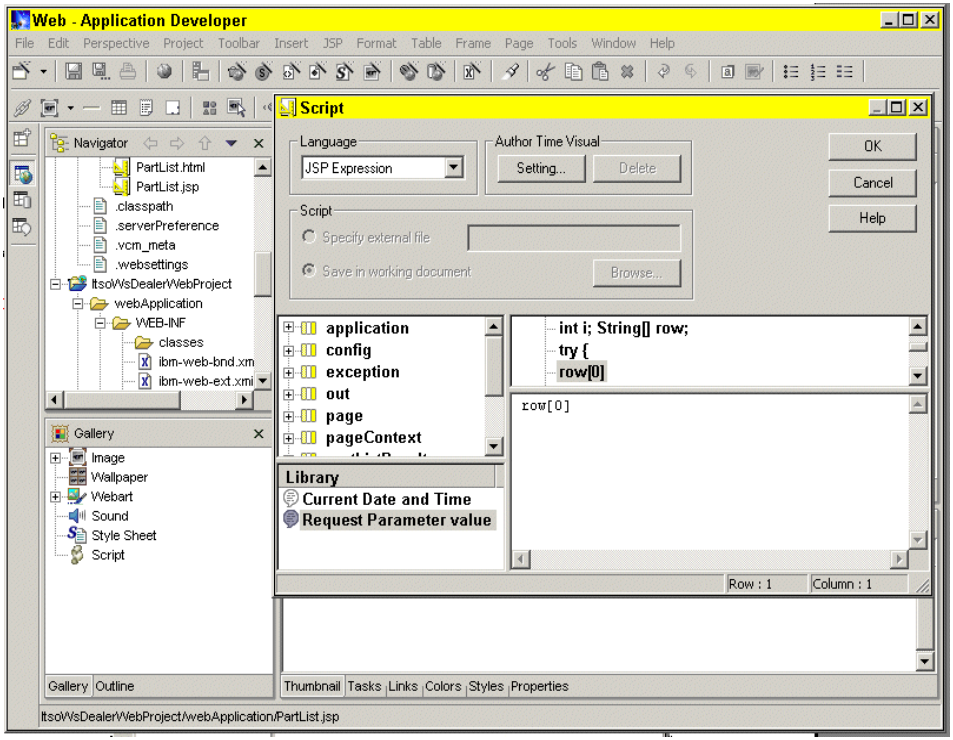


Figure 8-37 Entering the scripts for the table cells

This time you want to enter requests for parameter value. To do this select **JSP Expression** in the **Language** drop down list box and then you can enter the corresponding code in the left middle pane. From left to right you will need to enter `row[0]`, `row[1]`, `row[2]`, `row[3]` and `row[4]` for the individual cells.

After doing so, you need to enter the closing code for the loop (that you have opened in the script code and inserted between the table header row and the first row) and enter the exception handling code to protect against all exceptions, that might occur while processing the partListResult.

Since the loop was ‘inside’ the table, you need once again to switch to the Source view, where you can easily position the cursor between the end row tag and end table body tag and enter following code:

```
<%    }
   } catch (Exception e) {} %>
```

Figure 8-38 shows the code you need to enter and the place to insert it.

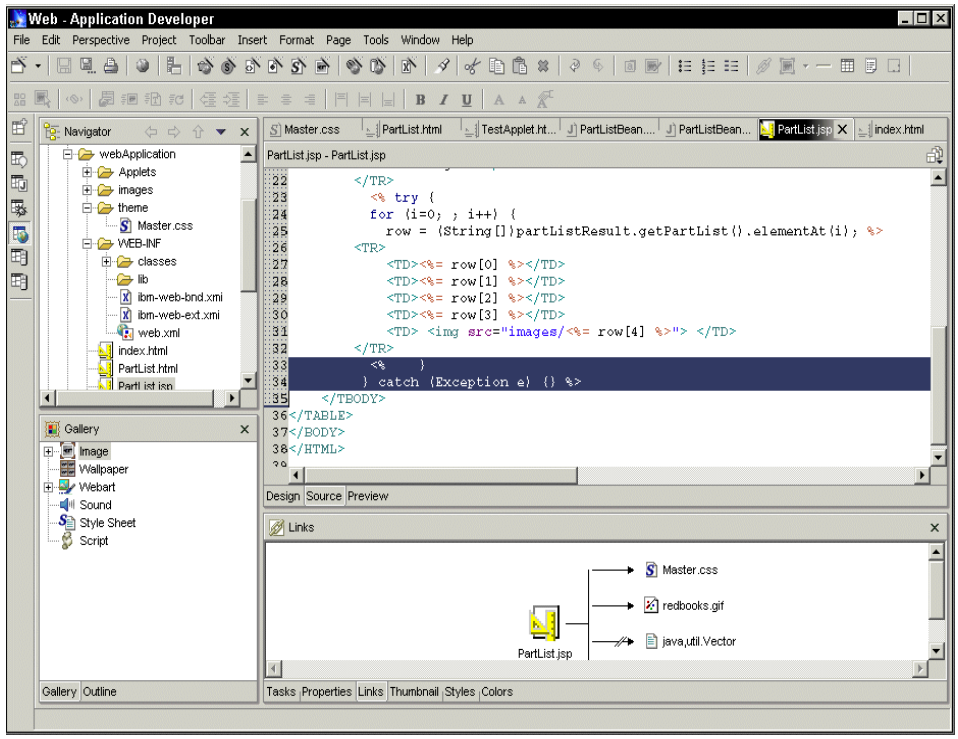


Figure 8-38 Closing the loop and protecting against exceptions

This completes the code for the JSP. You can test the visual aspects of it by selecting the Preview view in Page Designer. Figure 8-39 shows how it should look.

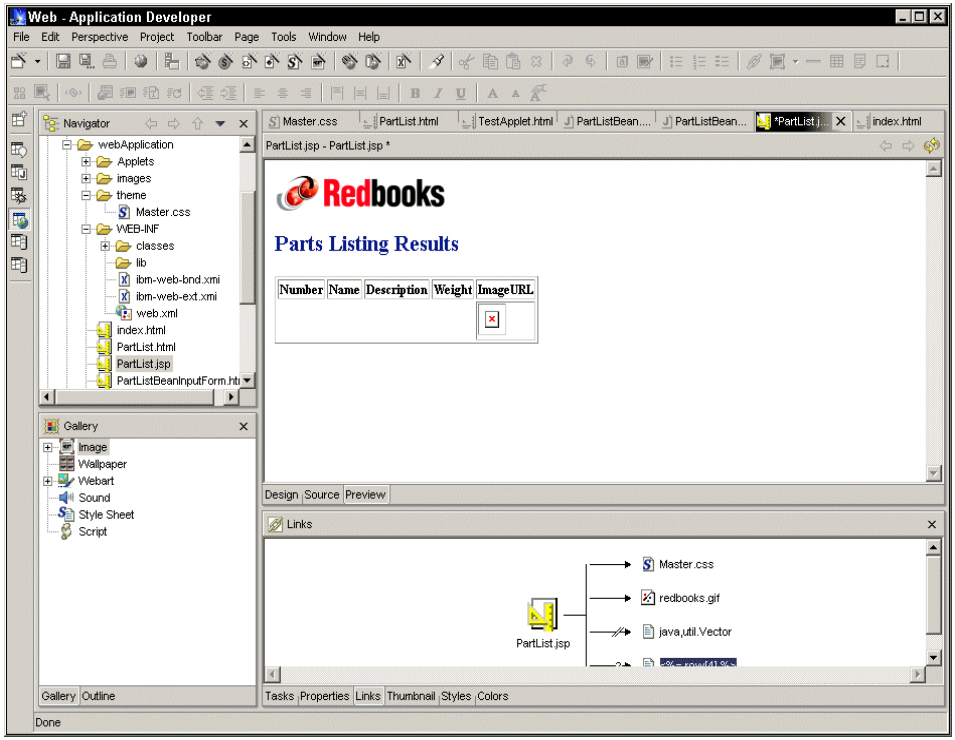


Figure 8-39 Preview of the finished JSP

Using taglibs

The *Sun Microsystems JSP 1.1 Specification* provides the ability to create custom JSP tags. Custom tags simplify complex actions and provide developers with greater control over page content. Custom tags are collected into a library (taglib). A tag library descriptor file (taglib.tld) is an XML document that provides information about the tag library, including the taglib short name, library description, and tag descriptions. Refer to the *Sun Microsystems JSP 1.1 Specification* for more details.

To use JSP 1.1 custom taglibs, you can import the taglib .tld and .jar files into your project to use. If you import these files, place them under the webApplication folder. You can also reference a .tld file using a URI.

We will not show you how to use custom taglibs in the context of our example, but you will see an example later in “Accessing a database using JSP taglib” on page 316.

8.4.3 Creating the controller servlet

Now all the pieces are in place for you to start implementing the MVC version of your servlet. This new servlet will have no application or presentation logic in it, but will instead use the Java bean and the JSP you have created to perform these tasks.

Note: The completed source code for the Controller servlet can be can found in:`..\Chapter8\PartList.java.`

You will add the new servlet to your ItsoWsDealerWebProject project and call it PartList. Start the Servlet wizard and enter `PartList` as the servlet name. Select the `init()` method stub to be generated and leave the rest of the first page as it was proposed by Application Developer.

Tip: Make sure that the right folder was selected before you started the Create Servlet dialog, (/ItsoWsDealerWebProject/source). If not, use the **Browse** button, next to the text field, to find and select it (Figure 8-40).

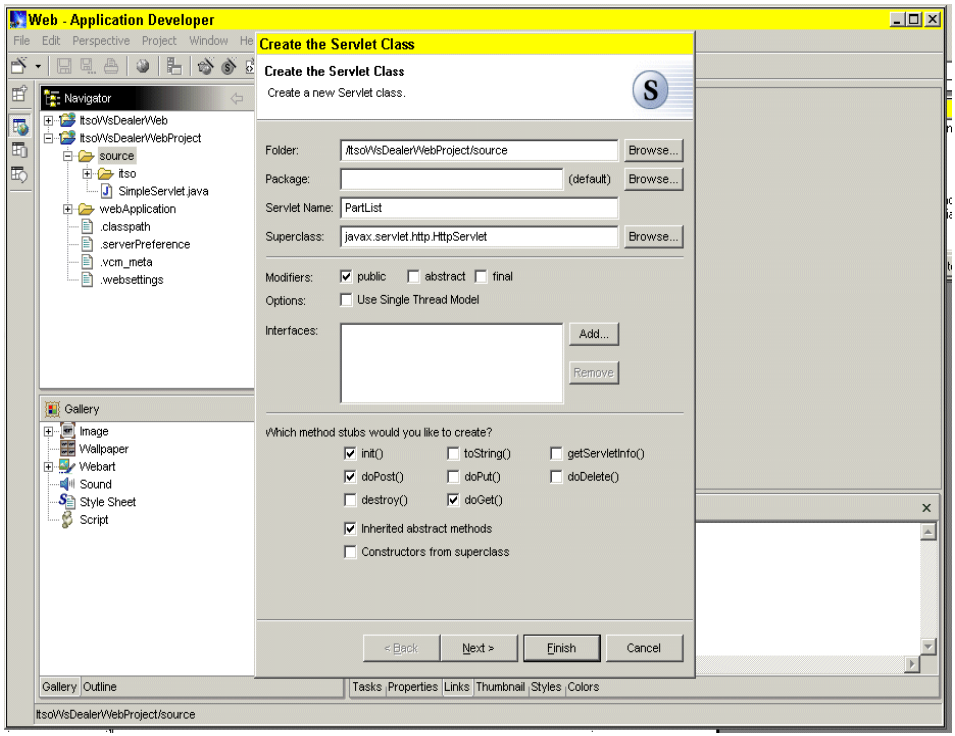


Figure 8-40 Creating the PartList servlet

On the next page (Figure 8-41) check if the **Add to web.xml** check box is selected.

As you will use the default display name and URL (`PartList`), so you do not need to make any changes to the defaults. The servlet will be added to the deployment web.xml descriptor.

Click **Finish**.

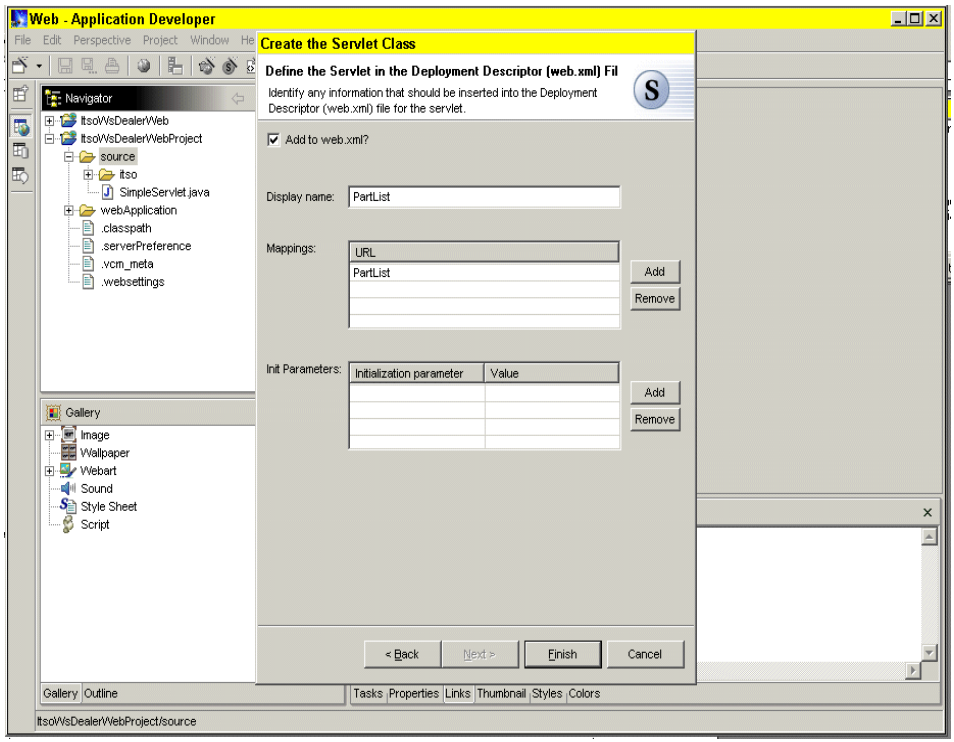


Figure 8-41 Deployment description for PartList servlet

The servlet will be added to the Web application and you should see it in the Navigator view. The Java editor will open on the servlet source code. You will see the skeleton code that you are already familiar with (Figure 8-42).

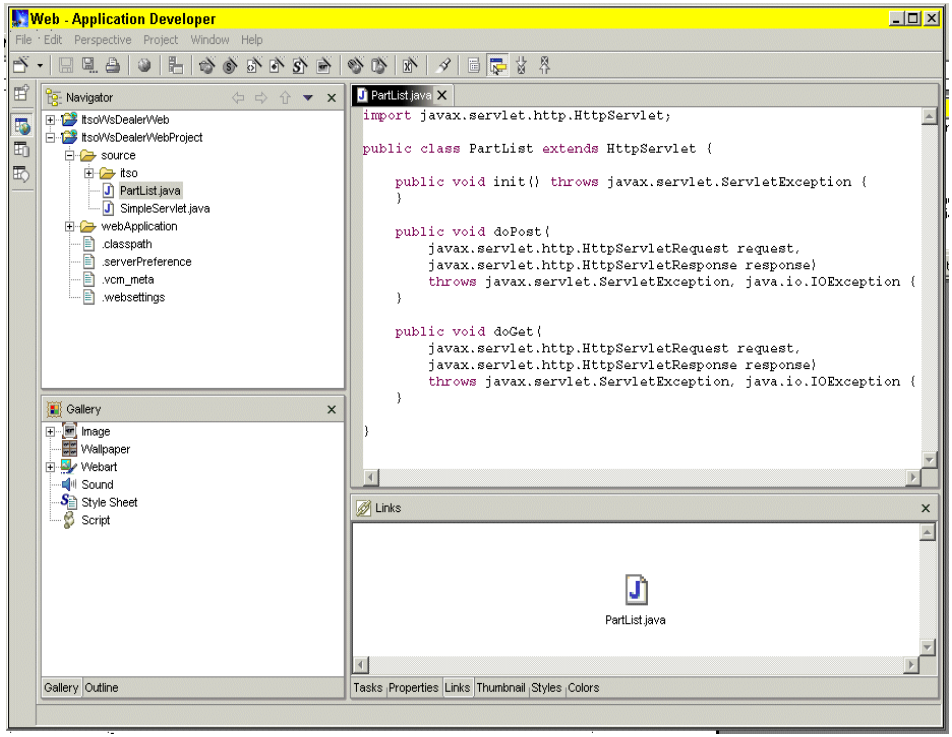


Figure 8-42 PartList servlet ready for editing

The steps that have to be completed now are basically the same as in “Creating a simple servlet” on page 207, but the way you will code the solution will be different.

First you need to add to the imports at the start of the PartList class source code. There will be two differences compared with list of imports that you have had for the SimpleServlet:

- As you do not access the database directly from the code of the Controller, you do not need the `import java.sql.*` anymore.
- As you now will use the PartListBean as a Model and it is defined in a package `itso.wsad.dealer.web`, you need to add this package to your imports.

  An alternative approach would have been to define all the Model, View and Controller classes in the same package.

Let us now look at the rest of the code. As you will see the end-result is not that different from the SimpleServlet code. This is reasonable, since both versions of the servlet implements the same behavior. However there will be a substantial change in the structure as you will introduce a certain level of independence between the individual layers of your application (Model, View and Controller).

You can leave the `init()` method unchanged, that is empty. Next you complete the code of the `doPost()` method in the same way you did in the SimpleServlet case. It will just call the `doGet()` method passing on the parameters.

The import statements and the `doPost()` code are shown in Figure 8-43.

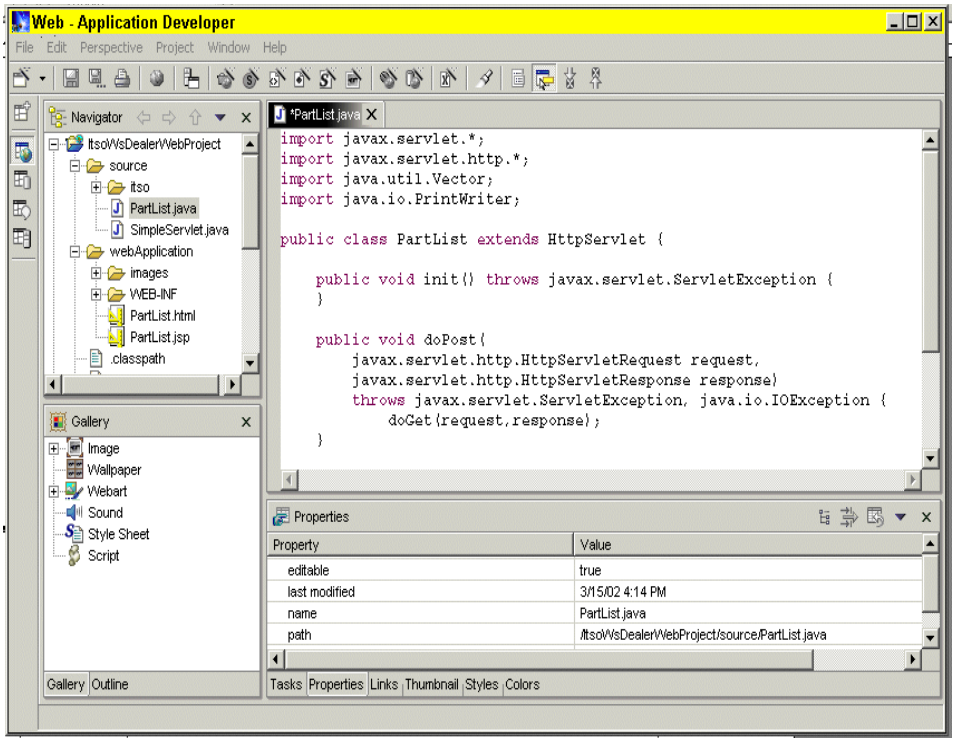


Figure 8-43 Import statements and doPost() method of the PartList servlet

Now you need to implement the `doGet()` method. In SimpleServlet this method did all the work. It accessed the database to get the list of parts and then created the resulting HTML page showing the results.

The new servlet will do the same thing, but it will use a Java bean (Model object) and a JSP (View object) to do the actual work.

Adding your Java bean as a Model

You added a Java bean to your Web application in “Creating the model Java bean” on page 224. You can now reference this bean in the code of the Controller servlet. It can be instantiated and you can ask it to perform the business logic functions, (in our case, the database access):

```
...
String partialName = request.getParameter("partialName");
PartListBean partListResult = new PartListBean();
partListResult.select(partialName);
...
```

Adding your JSP as a View

You created your View class (the JSP) in “Creating the view JSP” on page 230. Your Controller servlet can now reference it and use it to display the query results in HTML format:

```
...
String resultPage = "/PartList.jsp";
...
request.setAttribute("partListResult", partListResult);
//Forward the request to the next page
RequestDispatcher dispatch=request.getRequestDispatcher(resultPage);
dispatch.forward(request, response);
...
```

The complete code of the doGet() method in the PartList servlet class should look as in (Figure 8-44).

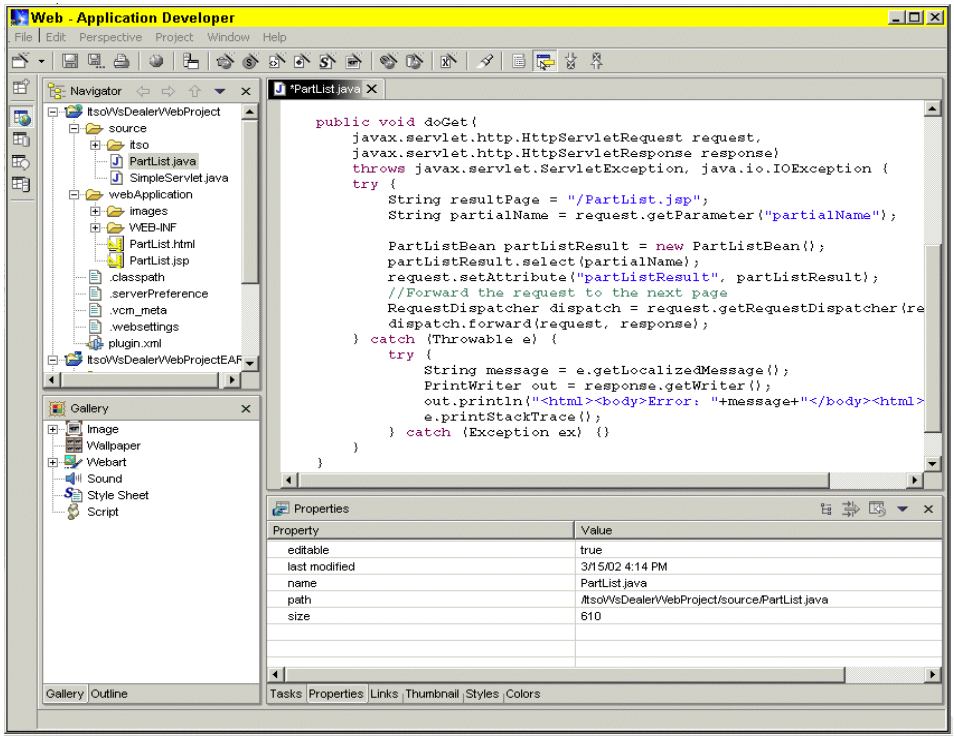


Figure 8-44 doGet() method from PartList servlet class

With this method completed, you have finished the last part of the code needed to implement the Part Listing Web application using the Model-View-Controller pattern. The finished application can now be tested.

8.5 Testing your Web application

To test the MVC version of the application you have to:

1. Change the action name in the `PartList.html` file (back to the **PartList** again instead of **SimpleServlet** you have used at the beginning of this chapter (Figure 8-45).



Figure 8-45 Changing the form's action back to PartList

2. Launch the `PartList.html` unit test in the local test environment by selecting **Run on Server** from the context menu of `PartList.html` in the Navigator view of the Web perspective Figure 8-46.

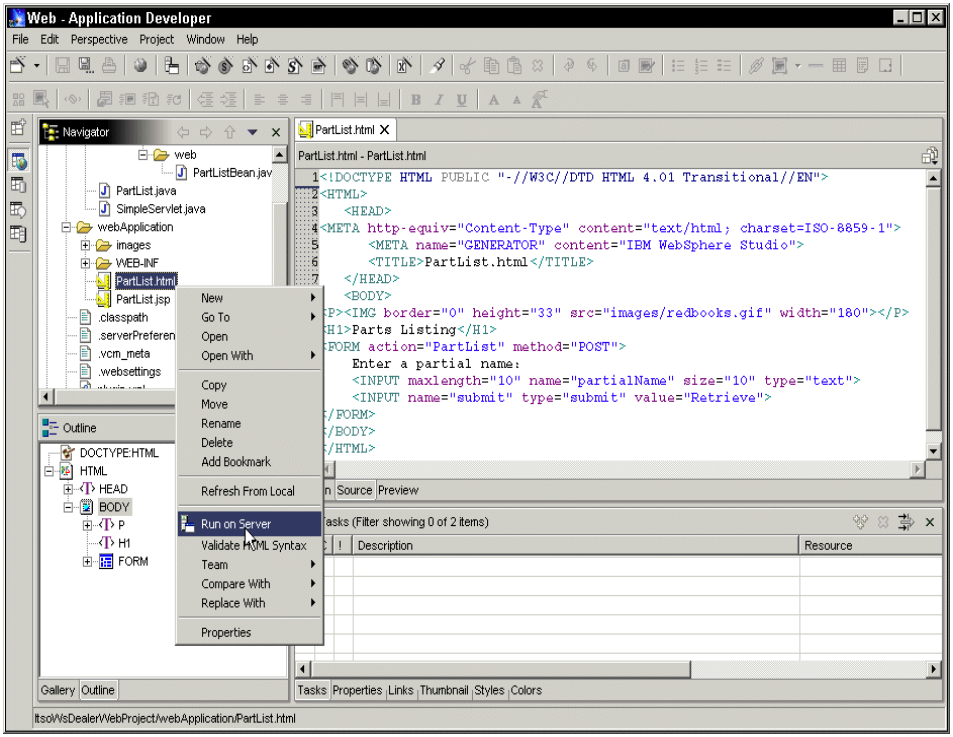


Figure 8-46 Starting the PartList servlet unit test

If required, the server will be synchronized and started at this point. (If that is the case Application Developer will switch to the Server perspective.) Once this processing is completed, a Web browser window will be opened in the top right pane of your workbench. It will display the Part.html page, which allows you to enter the Part search criteria.

3. Enter RR and click **Retrieve** (Figure 8-47).

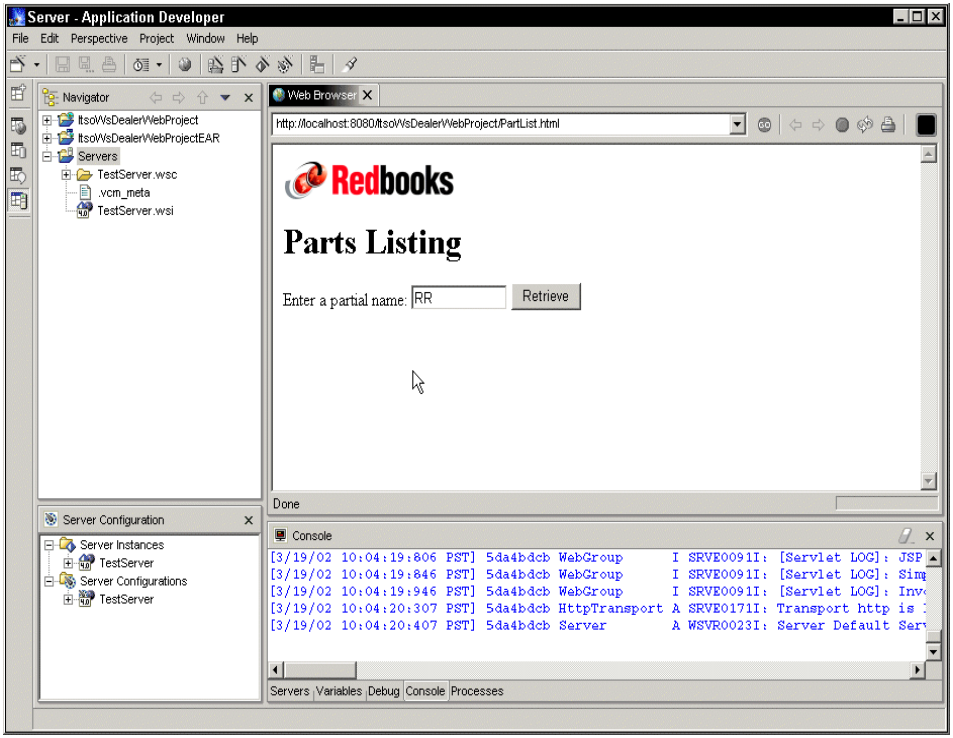


Figure 8-47 Starting from the PartList.html

You should see the same result as displayed for the SimpleServlet version of the application (Figure 8-48).

Note: The first time you run the application it will take some time before you see the result since Application Developer will need to compile the JSP and servlet code.

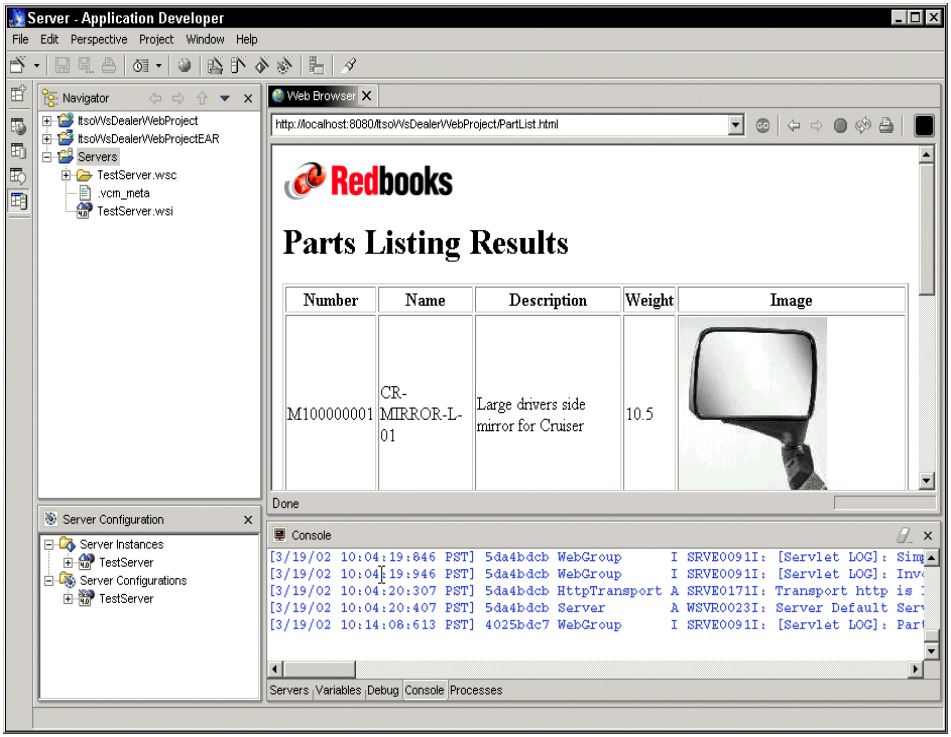


Figure 8-48 PartList servlet test result

If you see the result of your database query displayed in your browser as shown in Figure 8-48, you have succeeded to develop your MVC based Web application with Application Developer.

Part 3

Database applications

Part 3 adds a database accessing feature to the applications that were developed in Part 2.

© Copyright IBM Corp. 2002

9

Database connectivity

This chapter covers the following topics:

- JDBC overview
- Data Source versus direct connection
- Application Developer database operations
- XMI and DDL
- Data perspective
- Using DB Explorer
- Creating database objects

© Copyright IBM Corp. 2002

9.1 JDBC overview

JDBC, like ODBC, is based on the X/Open SQL call-level interface specifications, but unlike ODBC JDBC does not rely on various C features that don't fit well with the Java language. Using JDBC, you can make dynamic calls to databases from your Java applications or Java applets.

JCBC is vendor neutral and provides access to a wide range of relational databases, as well as to other tabular sources of data. It can even be used to get data from flat files or spreadsheets. This portability and versatility are the main attractions of using JDBC for database access in application programs. JDBC is especially suited for use in Web applications. Using the JDBC API you can connect to databases using standard network connections. Any modern Web browser is Java enabled so you don't have to worry about whether the client can handle the application or not.

Figure 9-1 shows the basic components of a JDBC connection. The JDBC API sends the SQL commands from the application to the JDBC driver manager, which in turn talks to the vendor specific driver that actually connects to the database.

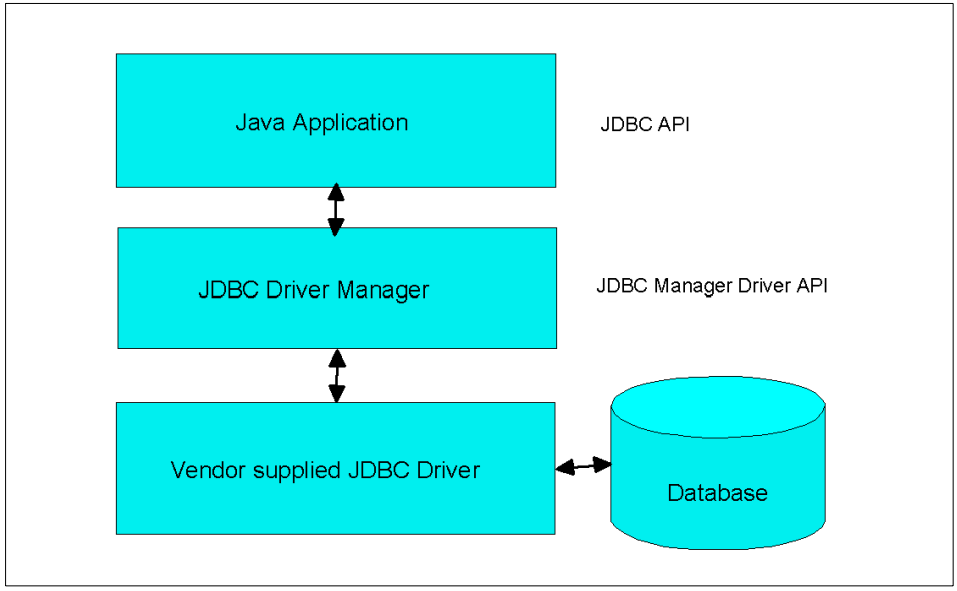


Figure 9-1 JDBC overview

9.2 Data source versus direct connection

In JDBC 1.0 the only way of establishing a database connection was by using the *driver manager* interface. This was expensive in terms of performance since a connection was created each time you needed to access the database from your program, thereby incurring a substantial processing overhead. In the JDBC 2.0 Standard Extension API an alternative way of handling database connections was introduced.

By using *data source objects* you have access to a pool of connections to a data source. Using *connection pooling* gives you the following advantages:

- It improves performance. Creating connections is expensive; a data source object creates a connection as soon as it is instantiated.
- It simplifies resource allocation. Resources are only allocated from the data source objects, and not at arbitrary places in the code.
- It simplifies connection calls. To get a connection in JDBC 1.0, you would need to call Class.forName() on the class name of the database driver, before making DriverManager calls.

Data source objects work as follows:

1. When a servlet or other client wants to use a connection, it looks up a data source object by name from a *JNDI* (Java Native Directory Interface) server.
2. The data source object then returns a connection to the client.
3. If the data source object has no more connections, it may ask the database manager for more connections (as long as it has not exceeded the maximum number of connections).
4. When the client has finished with the connection, it releases it.
5. The data source object then returns the connection to the available pool.

Important: Because of the advantages of connection pooling, using data source objects is the preferred method of handling database connections in Web applications. The WebSphere Application Server has full support for connection pooling and for registering data sources via JNDI.

If you use the Create database Web pages wizard, described in detail in “Generate Web pages from SQL queries” on page 304, you have the option of generating code to use either a driver manager connection or a data source connection.

If you use the driver manager connection the following method to initialize the connection is generated:

```
protected void initConnectionSpec() throws SQLException {
      connectionSpec = new DBConnectionSpec();
      connectionSpec.setUsername(getUsername());
      connectionSpec.setPassword(getPassword());
      connectionSpec.setDriverName(getDriverName());
      connectionSpec.setUrl(getUrl());
}
```

If you instead use the data source method, the initConnectionSpec() method will look like:

```
protected void initConnectionSpec() throws SQLException {
      connectionSpec = new DBConnectionSpec();
      connectionSpec.setUsername(getUsername());
      connectionSpec.setPassword(getPassword());
      connectionSpec.setDataSourceName(getDataSourceName());
  }
```

Tip: Both the driver and data source information are stored in the deployment information file for the project, (web.xml). Not hardcoding the driver and data source names makes it easier to make changes.

As you can see in from these code snippets, the DBConnectionSpec class encapsulates most of the differences between direct connections and data sources, making it easy to change your code from one method to the other.

To use a data source you need to configure the server to recognize it. As an example, if you wanted to register a data source for the sample ITSOWSAD database, you would follow the following steps.

Open the Server perspective and edit the server configuration for the project. Go to the Data source tab, select the Db2JdbcDriver from the top pane and click **Add** in the data source pane. The Data Source dialog will be displayed (Figure 9-2).

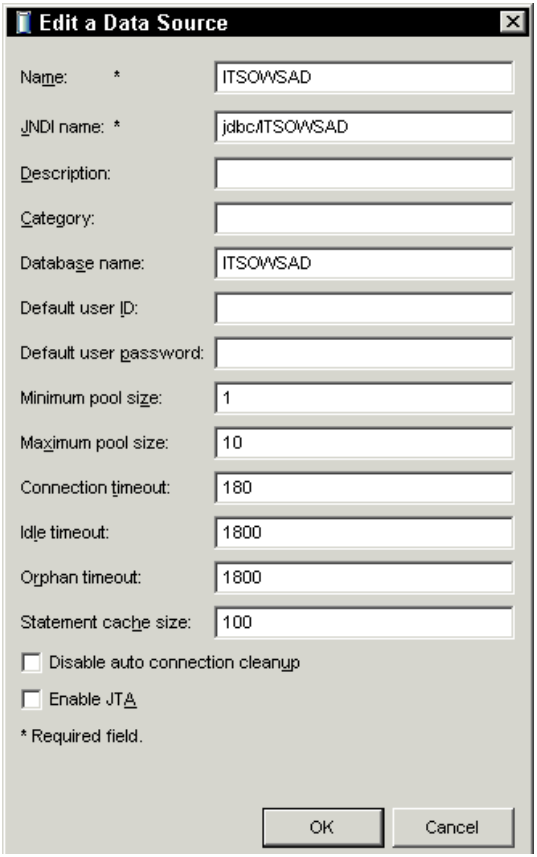


Figure 9-2 Adding a datasource

Enter *ITSOWSAD* as the **Name**, `jdbc/ITSOWSAD` as the **JNDI name** and *ITSOWSAD* as the **Database name**. Click **OK** and press Ctrl-S to save the configuration. If the server was running it will have to be restarted before running the application that uses the data source.

9.3 Application Developer database operations

Application Developer provides a number of features that make it easier to work with relational databases in your projects.

- Ability to import and use existing database models.
- Ability to create your own database objects and generate DDL for the target database.

- Ability to generate XML schemas from database models.
- SQL Wizard and SQL Query Builder to interactively build and execute SQL queries from an imported database model or via an active connection.
- Ability to generate Web pages and supporting Java classes based on existing or new SQL queries.
- Ability to access database API from JavaServer pages using either Java Beans or JSP tags.

These features will be discussed in more detail in the following chapters.

9.4 XMI and DDL

XMI (XML Metadata Interchange) is an OMG, (Object Management Group), standard format for exchanging metadata information. Application Developer uses the XMI format to store all local descriptors of databases, tables and schemas. The content of the XMI files can be viewed and edited using tailored editors. When you import an existing database model, it will be stored in XMI format.

DDL (Data Definition Language) is a format used by relational database systems to store information about how to create of database objects. Application Developer allows you to generate DDL from an XMI file and vice versa.

9.5 Data perspective

The Data perspective is used to access the features previously described. The various components of this perspective are described in some detail in “Data perspective” on page 85. There are three main views in the Data Perspective:

- DB Explorer view
- Data view
- Navigator view

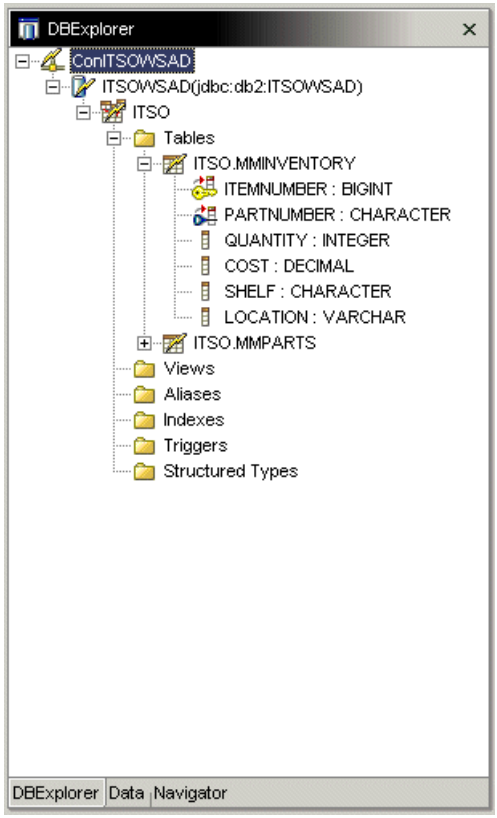


Figure 9-3 DB Explorer view

The *DB Explorer view* shows active connections to databases and allows you to create new connections. In Figure 9-3 you can see an active connection to the ITSOWSAD DB2 database.

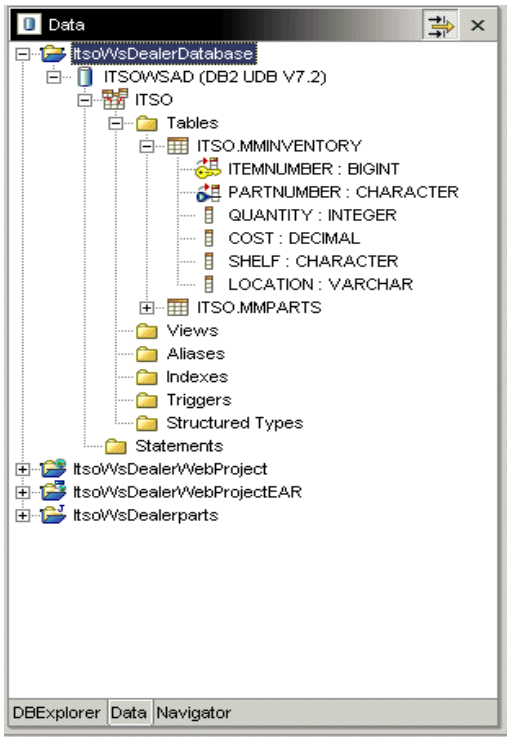


Figure 9-4 Data view

The *Data view* shows the database models that currently exist in Application Developer. These are either imported via the DB Explorer view or created within the workbench. The example shows that we have imported the datamodel that we connected to using the DB Explorer view.

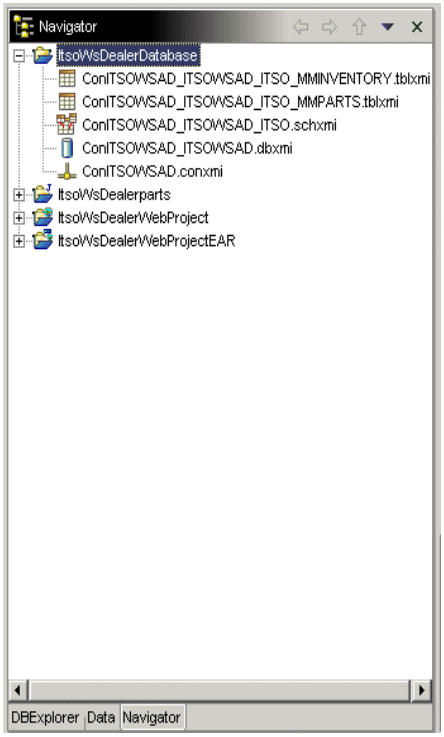


Figure 9-5 Navigator view

In the *Navigator view* you can see the local descriptor files, (.xmi files), that represent the database objects. Each of them has an editor associated with it. Double-clicking on the file will bring up the appropriate editor for the type of object that is described, which could be either a database, a schema or a table.

In the next section we will look in more detail at how you can work with the DB Explorer view.

9.6 Using DB Explorer

You can use DB Explorer to connect to existing databases and view their designs. The designs can be imported into Application Developer and used in your applications. DB Explorer allows you to filter the designs that are returned to only show a subset of schemas or tables. You can also use DB Explorer to generate DDL files and XML schemas.

Important: The DB Explorer view is read-only. Before you can edit any database objects you have to import them into a Application Developer project.

To access DB Explorer, activate the Data perspective of your project and select the DB Explorer view in the left pane.

9.6.1 Creating a database connection

Note: The examples in this section assume that you have created and populated the DB2 tables as described in Appendix C, “Additional material” on page 675.

To view the definition of an existing database you first have to create a JDBC connection to it. To create a new connection, first make sure you are in the DB Explorer view. Then right-click anywhere in the view and select **New Connection** to display the Connection wizard (Figure 9-6).

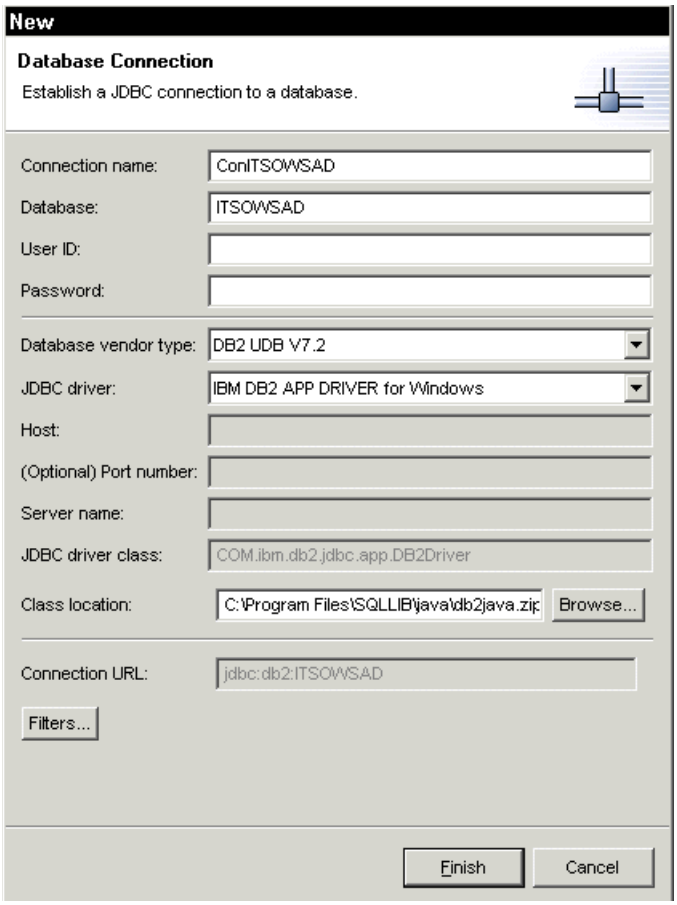


Figure 9-6 Creating a JDBC connection

You need to provide a unique name for the connection, a userid and password if required, and the type of database your are connecting to. You also need to specify which JDBC driver that should be used. There are two predefined JDBC drivers to select from for Windows:

- IBM DB2 App Driver (`COM.ibm.db2.jdbc.app.DB2Driver`) for connection to local databases or remote databases defined locally with the DB2 Client Configuration Assistant.
- IBM DB2 Net Driver (`COM.ibm.db2.jdbc.net.DB2Driver`) for connection to remote DB2 databases.

If you don't want to use one of the predefined drivers, you can select `Other DB2 driver` from the JDBC driver drop-down, and fill in the **JDBC driver class** field with the driver name.

if you want to limit the schemas or tables returned, click **Filters** (Figure 9-7).

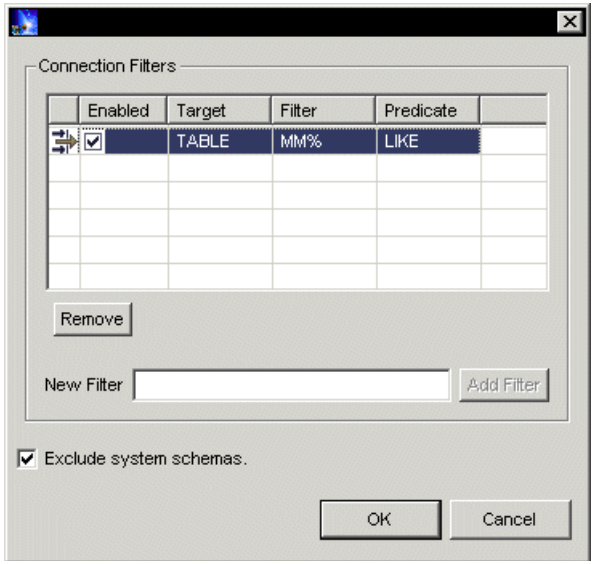


Figure 9-7 Creating a JDBC connection - apply table filter

In the preceding example we are limiting the selection to tables with names starting with MM. To create the filter type `MM%` in the **New Filter** entry field and click **Add Filter**. You can modify the **Predicate** by clicking on that cell in the table. If you wanted to filter on schema name, you would change the **Target** column from `TABLE` to `SCHEMA`. After clicking **OK** and **Finish** in the previous window, the connection is created and a new entry is added to the DB Explorer view. You can expand the new node to see the schemas and tables that are now available for use (Figure 9-8).

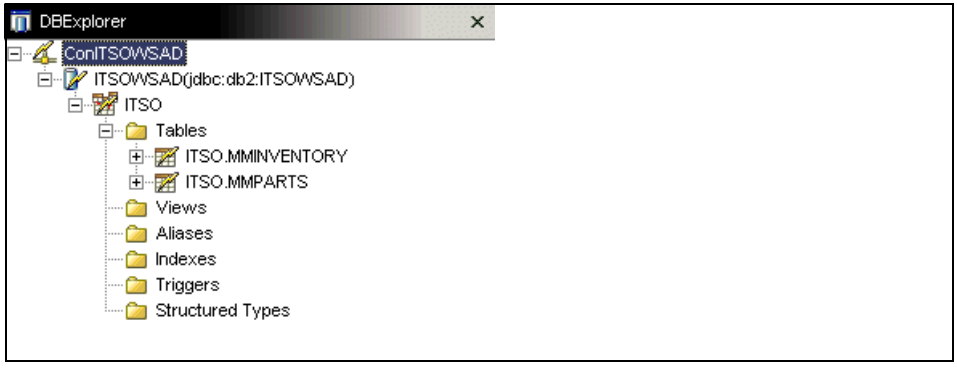


Figure 9-8 DB Explorer view of ITSOWSAD database with table filter applied

9.6.2 Import database objects

In the DB Explorer view you can browse the tables and columns, but before you can actually use them in your application, you need to import them into a folder in your project.

To have somewhere to store the imported objects you should now create a new project. To do so select **File—>New**. Then select **Simple** and **Project** from the New Project dialog and click **Next**. (A simple project is a generic project that contains files and folders.) Enter `ItsoWsDealerDatabase` as the project name.

We will now show you how to import the ITSOWSAD database into the new project.

Right-click the connection that you just created in the DB Explorer view and select **Import to folder**.

Specify the folder to import to and click **Finish** (Figure 9-9).

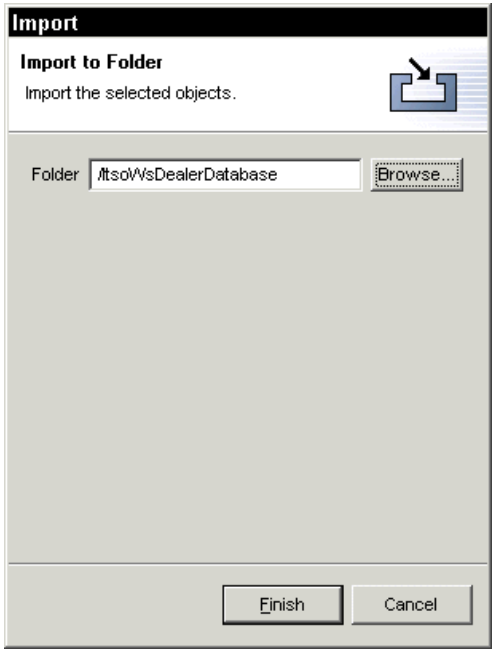


Figure 9-9 Import database objects

You can now switch to the Data view and expand the ITSOWSAD node. The same database objects will be shown, but you can now open editors on the them to view and modify their definitions.

If you switch to the Navigator view you will notice that a number of XMI files have been created for the database objects. (XMI is an open information interchange model that allows developers who work with object technology to exchange programming data over the Internet in a standardized way.) If you double-click on one of these files, the appropriate object editor will open up. If you want to see the xmi source, you can right-click on any of the files and select **Open With—>Default Text Editor**.

9.6.3 Generate DDL and XML Schema files

Application Developer allows you to generate DDL files and XML Schemas for database objects. To generate a DDL file, right-click the database object in the Data or DB Explorer view and select **Generate DDL** (Figure 9-10). You can generate DDL for the database, for a schema or for individual tables.

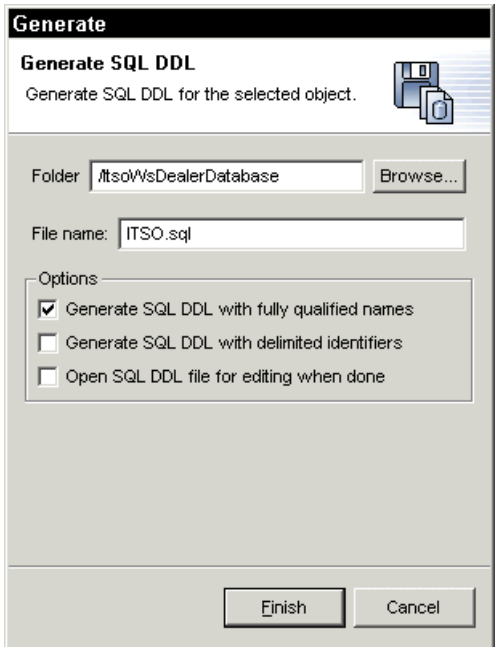


Figure 9-10 Generate DDL for a database object

Enter the name of the folder where you want the generated .sql file to be stored, select options for the generation and whether you want to open a text editor on the generated file. If you elect not to open the editor, you will need to switch to the Navigator view to see the generated file.

XML schemas can be generated for tables. To generate an XML schema for a table you must already have imported it into a folder and be in the Data view. Right-click the table and select **Generate XML Schema** (Figure 9-11).

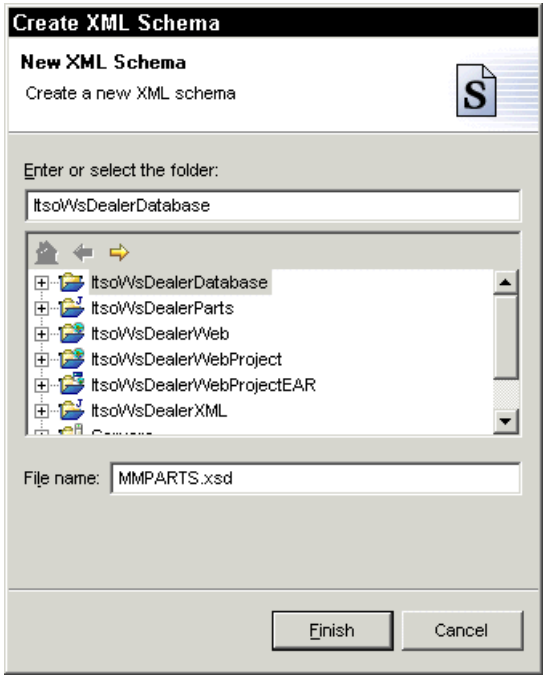


Figure 9-11 Generate XML schema for database table

When you click **Finish** the schema file (with extension .xsd) will be created. You can edit it using the Application Developer XML editor like any other XML file. If you want, you can make changes to the XML file and generate a new DDL file from it.

Now we will look at how you can use Application Developer to create new database objects.

9.7 Creating database objects

Application Developer provides support for creating new databases, new schemas and new tables.

To create a new database you need to have a project. If you haven't already done so, you should now create a new project called ItsoWsDealerDatabase. This project should be of type *Simple*. (A simple project is a generic project that contains files and folders.) To create database objects you need to switch to the Data perspective and open the Data view.

To create a database you right-click the ItsoWsDealerDatabase project and select **New—>New Database**. The Database creation dialog is displayed (Figure 9-12).

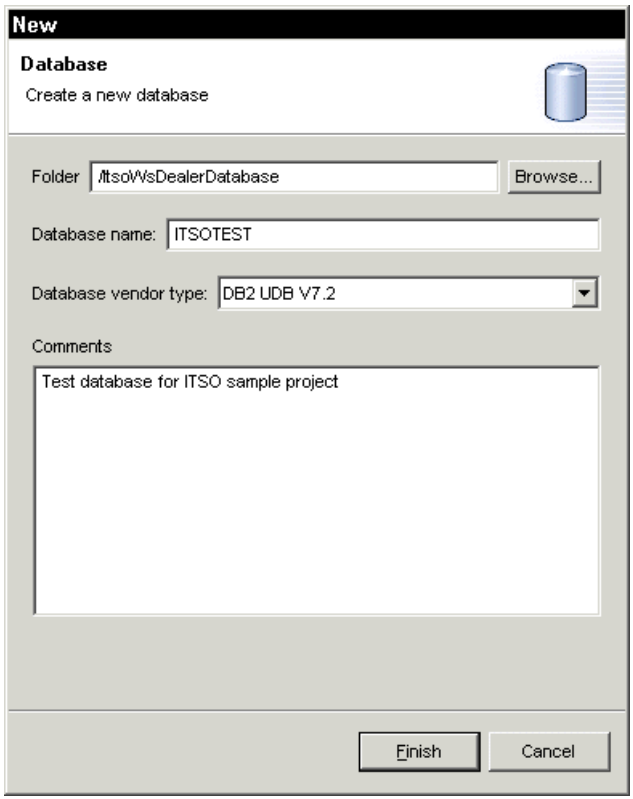


Figure 9-12 Database creation dialog

Here you specify the name of the new database and the vendor type. When you later generate the database DDL it will conform to the database type you select here.

Important: Database objects created within Application Developer are not automatically created in the database system. You need to export the DDL and use the appropriate database tool to create the objects.

Database schemas are a way of providing a logical classification of objects in the database. Some of the objects that a schema may contain include tables, views, aliases, indexes, triggers, and structured types. The support for schemas varies between database types, some require them and some have no support for them. The schema options available to you will depend on the database type that you chose when the database was created. If the database type doesn't support schemas at all, this option will not be available and the tables and other objects will be created directly under the database node.

To add a schema to the database, right-click the database created in the previous step and select **New—>New schema**. The Schema creation dialog is displayed (Figure 9-13).

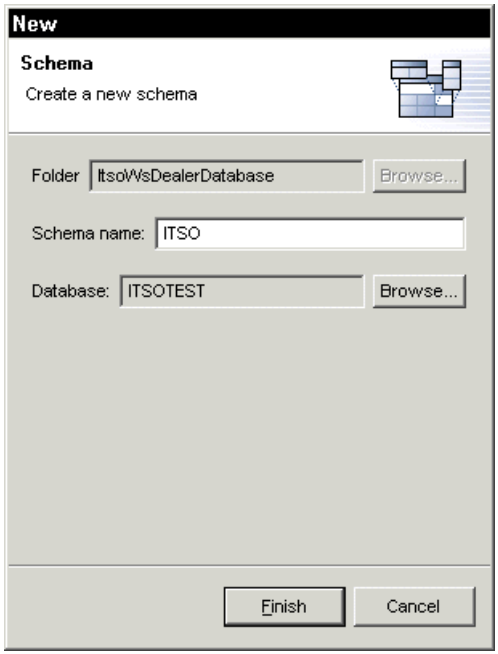


Figure 9-13 Schema creation dialog

Select a name for the schema and click **Finish** to create it.

If you expand the new schema in the Navigator view, you will see the types of objects that can be added to it.

- Tables
- Views
- Aliases
- Indexes
- Triggers

- Structured types

Restriction: In the current release of Application Developer only tables can be created. The other types of objects are not yet supported.

We will now look at how to create a new table in the schema. Application Developer provides a wizard for defining table columns as well as primary and foreign keys. To create a table you right-click the schema created in the previous section and select **New—>New Table**. The Create table wizard is displayed (Figure 9-14).

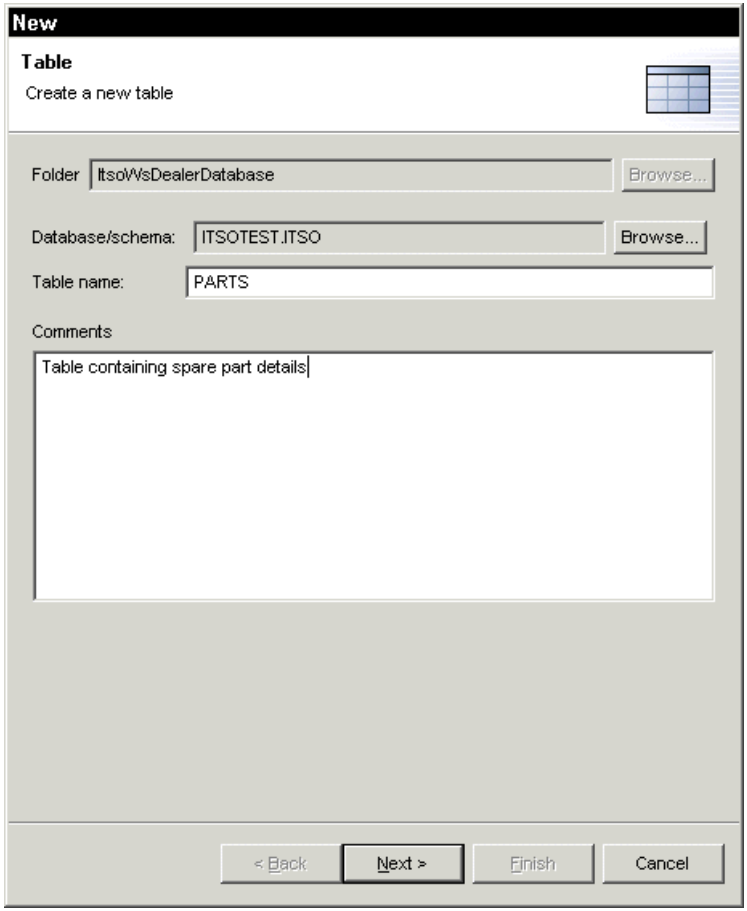


Figure 9-14 Table creation wizard - table name

Here you give the table a name and an optional comment. On the next page you define the columns of the table (Figure 9-15).

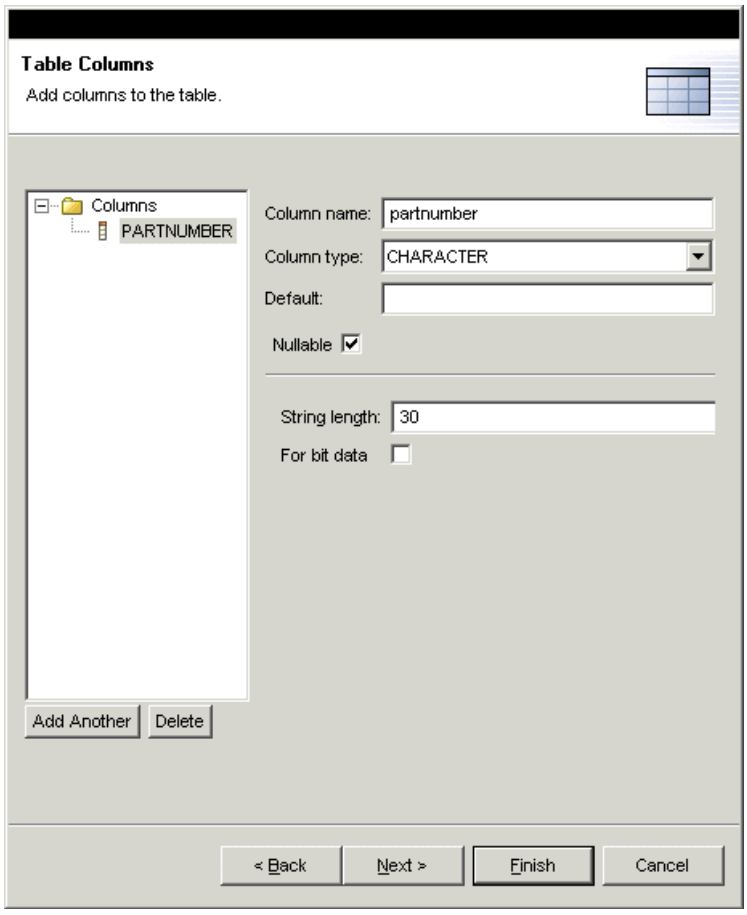


Figure 9-15 Table creation wizard - columns

Click **Add Another** to add a column to the table and define the column properties. The exact properties available will depend on the database type. For more information about the properties available, you need to consult the documentation provided by the database vendor.

The next page of the wizard lets you define the primary key of the table (Figure 9-16).

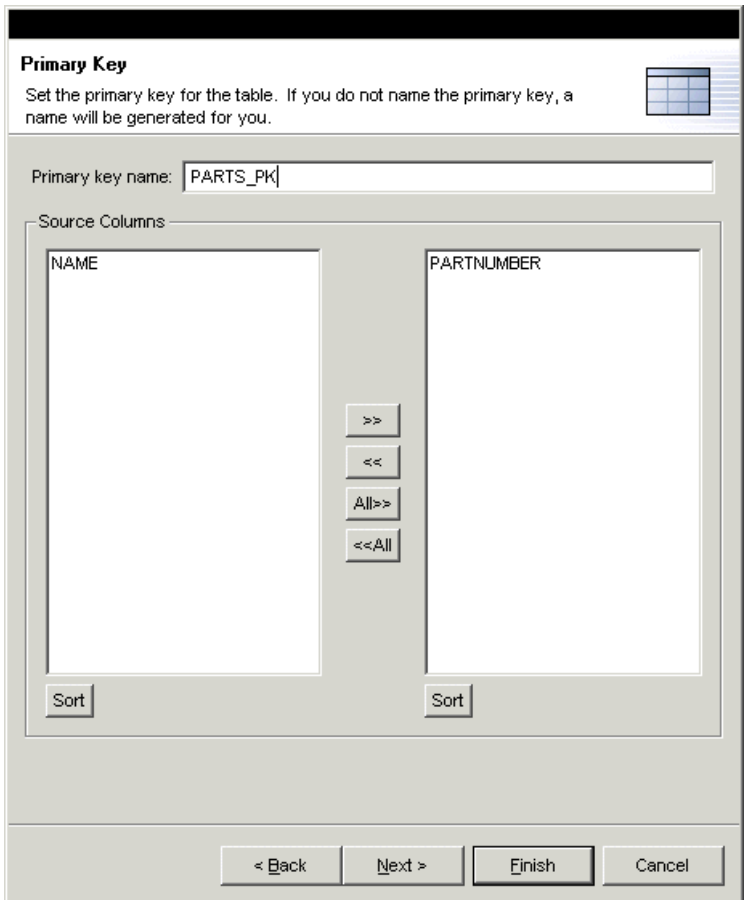


Figure 9-16 Table creation wizard - primary key

You select the items you want from the Source Columns and add them to the primary key by clicking the **>>** button.

On the final page of the wizard you can define any foreign key constraints that you want to apply. This assumes that you have already created another table with a primary key (Figure 9-17).

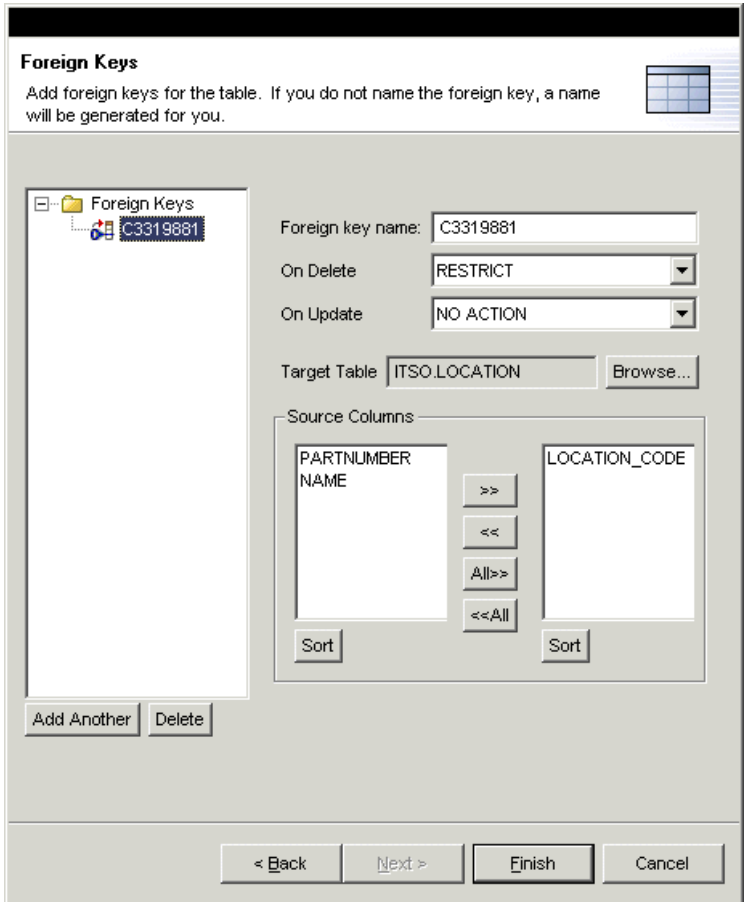


Figure 9-17 Table creation wizard - foreign keys

Clicking **Finish** will create the table and keys as defined.

If you want, you can generate the DDL for the table you've just created. To do so, right-click the table in the Data view and select **Generate DDL** (Figure 9-18).

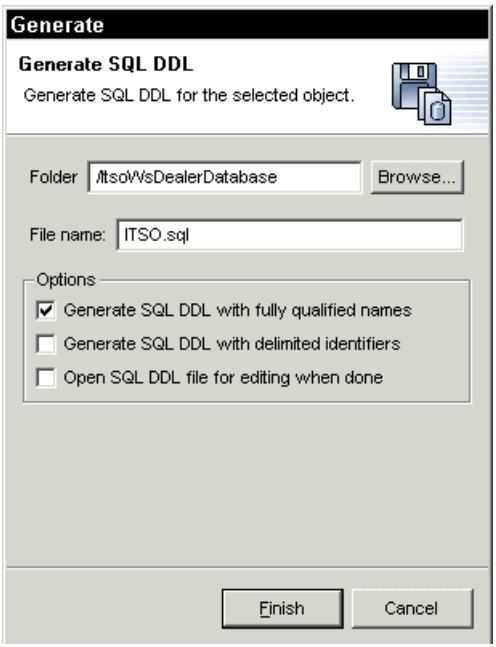


Figure 9-18 Generate DDL dialog

The options available are to create the DDL with or without the schema name, whether to place delimiters around identifiers or not, and whether to open an editor on the generated file.

You can use the generated DDL to create the table in the database system with the help of the appropriate tool provided by the database vendor.

10

Using SQL Wizard and SQL Query Builder

This chapter discusses ways of creating SQL statements in Application Developer. The following tools will be described:

- SQL Statement Wizard
- SQL Query Builder

© Copyright IBM Corp. 2002

10.1 Using the SQL Wizard

There are two alternative ways of creating an SQL statement in Application Developer. In this section we will describe how to use the *SQL Wizard* and in the following section we will show how to do the same thing using the *SQL Query Builder*.

For our example we will be developing a `SELECT` statement against the sample `ITSOWSAD` database. We would like to select all parts from the `MMPARTS` table with a part number beginning with “M” and that have a weight greater than 2. We would also like to show the quantity on hand for these parts.

We will now show you how to create an SQL statement using the wizard.

Highlight the database project in which you want to store the new statement and and select **File—>New—>Other**. Then select **Data** and **SQL Statement** from the New dialog.

Click **Next** to open the Create a New SQL Statement wizard (Figure 10-1).

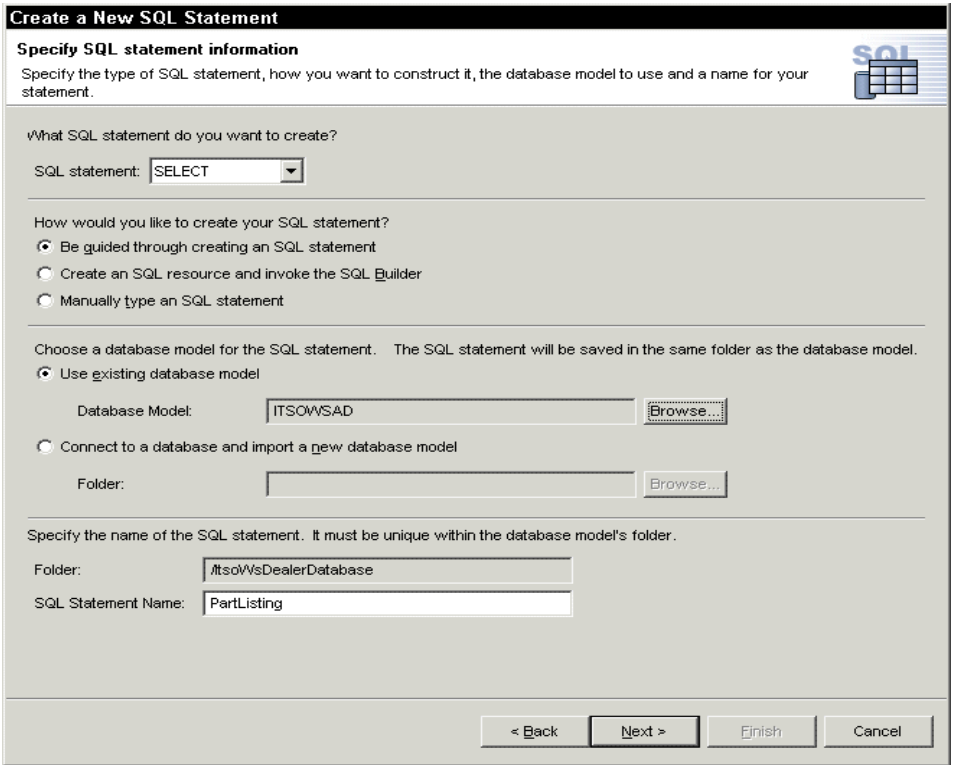

Create a New SQL Statement
Specify SQL statement information
Specify the type of SQL statement, how you want to construct it, the database model to use and a name for your statement.
SQL
What SQL statement do you want to create?
SQL statement: SELECT
How would you like to create your SQL statement?
Be guided through creating an SQL statement
Create an SQL resource and invoke the SQL Builder
Manually type an SQL statement
Choose a database model for the SQL statement. The SQL statement will be saved in the same folder as the database model.
Use existing database model
Database Model: ITSOWSAD
Browse...
Connect to a database and import a new database model
Folder:
Browse...
Specify the name of the SQL statement. It must be unique within the database model's folder.
Folder: /ItsoWsDealerDatabase
SQL Statement Name: PartListing
< Back
Next >
Finish
Cancel

Figure 10-1 SQL Statement Wizard - specify statement information

On the first page you select the type of statement you want to create and where to find the database model to use for building the statement. There are two ways to get the database model. You can either use an existing one or import a new one. In this case we already have the database model imported into the workbench so we select the **Use existing database model** radio button. Click the **Browse** button to locate the ITSOWSAD model in the workbench.

On the second page of the SQL Statement Wizard you build your SQL statement by selecting tables, columns and adding joins and conditions. First we identify the tables that should be included in the query (Figure 10-2).

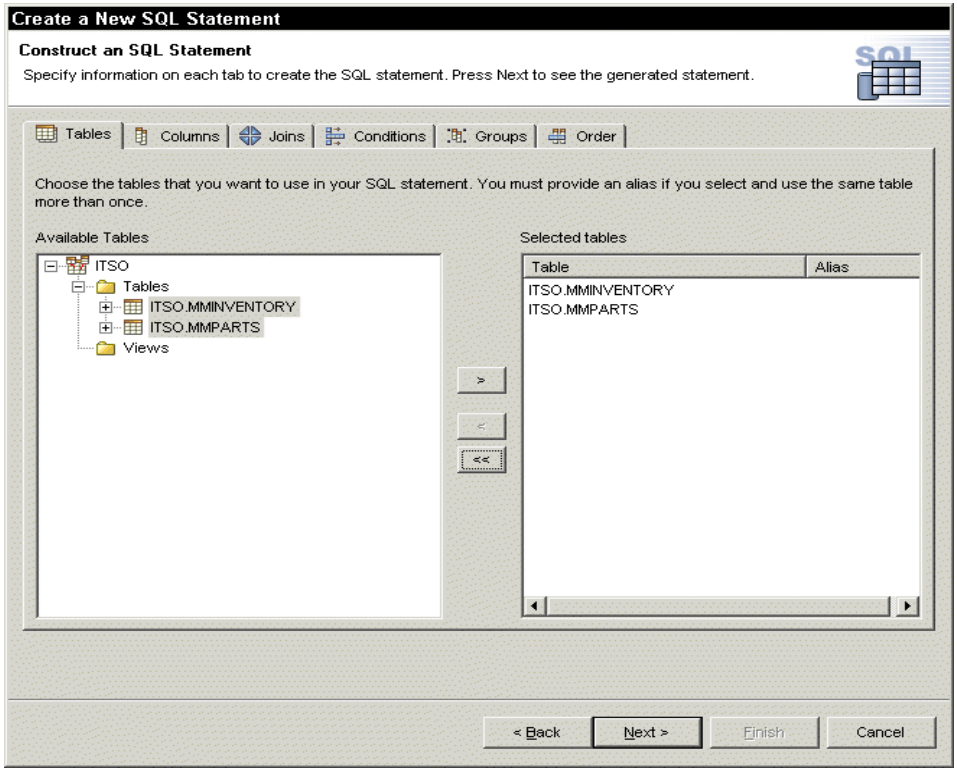


Figure 10-2 SQL Statement Wizard - add tables

You highlight the tables you want in the left pane and use the **>>** button to include them. For our example we need the `MMPARTS` and `MMINVENTORY` tables. On the *Columns* tab you select the columns from the two tables that should be included in the query (Figure 10-3).

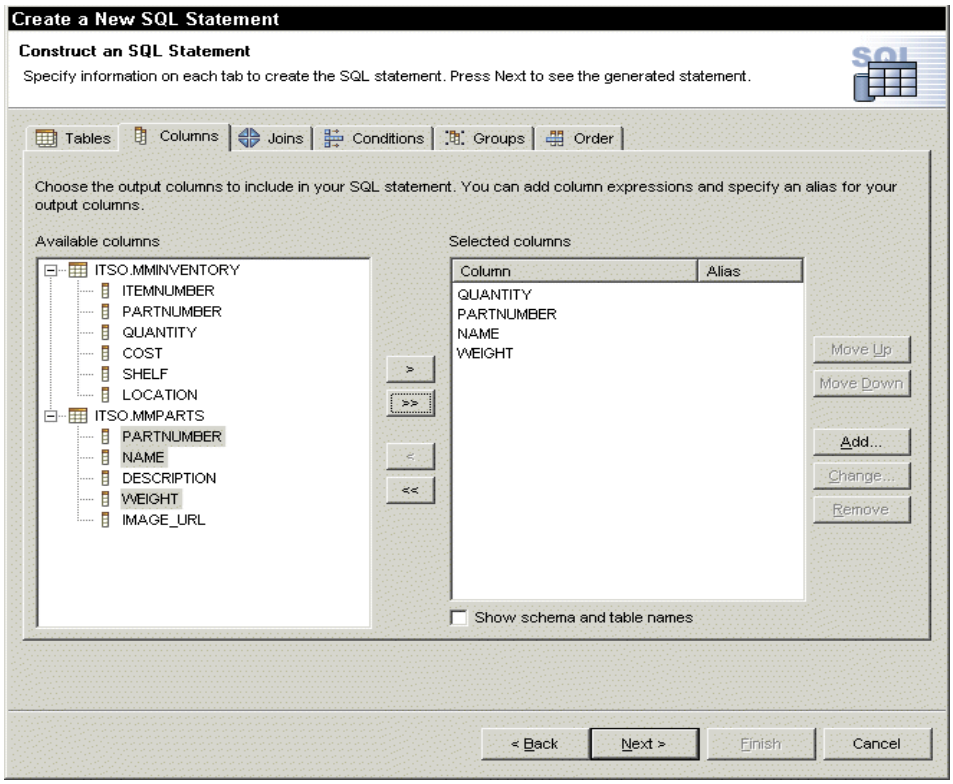


Figure 10-3 SQL Statement Wizard - add columns

`PARTNUMBER`, `NAME` and `WEIGHT` from the `MMPARTS` table and `QUANTITY` from the `MMINVENTORY` table are selected and moved across to the right pane.

Next you need to specify the join column between the two tables on the *Join* **tab**. This is done by selecting the column from one table and dragging it to the corresponding column of the other table. In our case we link `MMPARTS.PARTNUMBER` to `MMINVENTORY.PARTNUMBER`. When the join is completed a connection symbol is displayed (Figure 10-4).

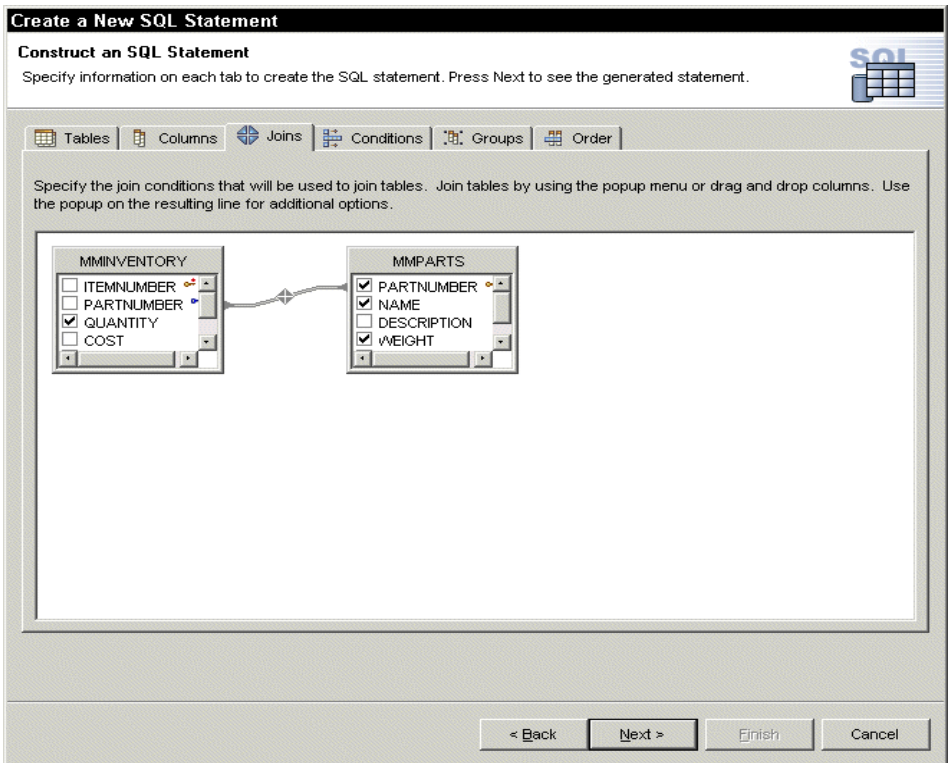


Figure 10-4 SQL Statement Wizard - add joins

The *Conditions* tab is used to define the restrictions on the `SELECT` statement. Each condition will translate into an `SQL WHERE` clause (Figure 10-5).

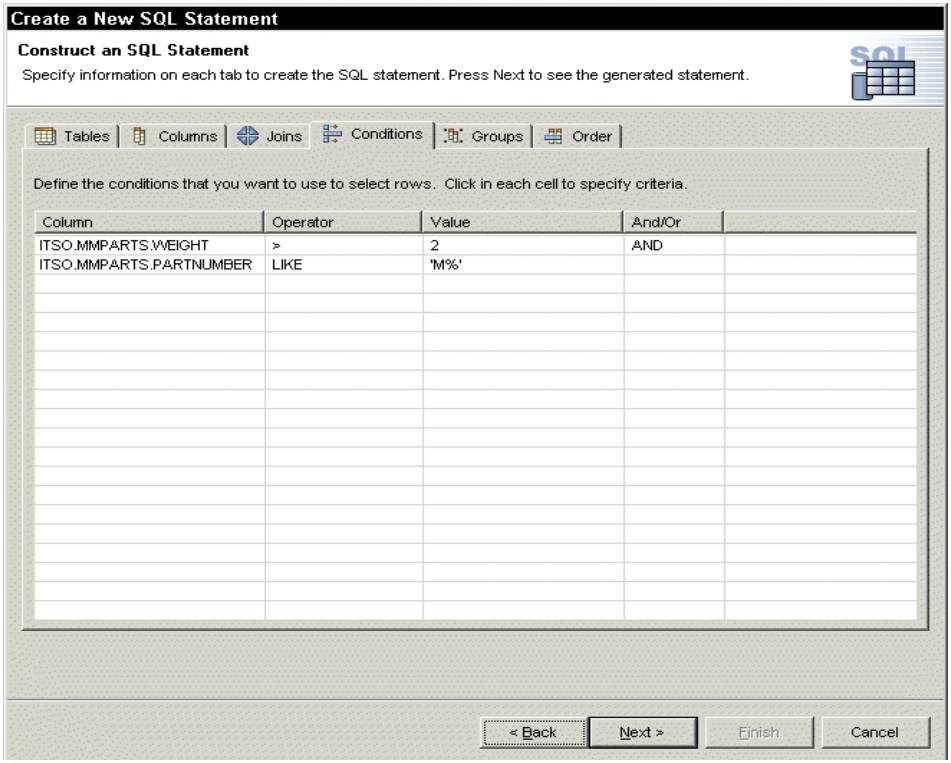


Figure 10-5 SQL Statement Wizard - add conditions

In a real-life situation you might not want to hardcode in the limit on weight as 2, but instead leave it as a host variable. To do this replace the `2` in the value column with `:weight`.

Tip: If you need to enter more than one condition, you must put in the `AND` or the `OR` element before the next row in the table becomes editable.

On the next two tabs you can enter information regarding grouping (`GROUP BY`) and sorting of rows (`ORDER BY`).

Once you have finished building the statement you can click **Next** to see how the generated SQL statement looks (Figure 10-6).

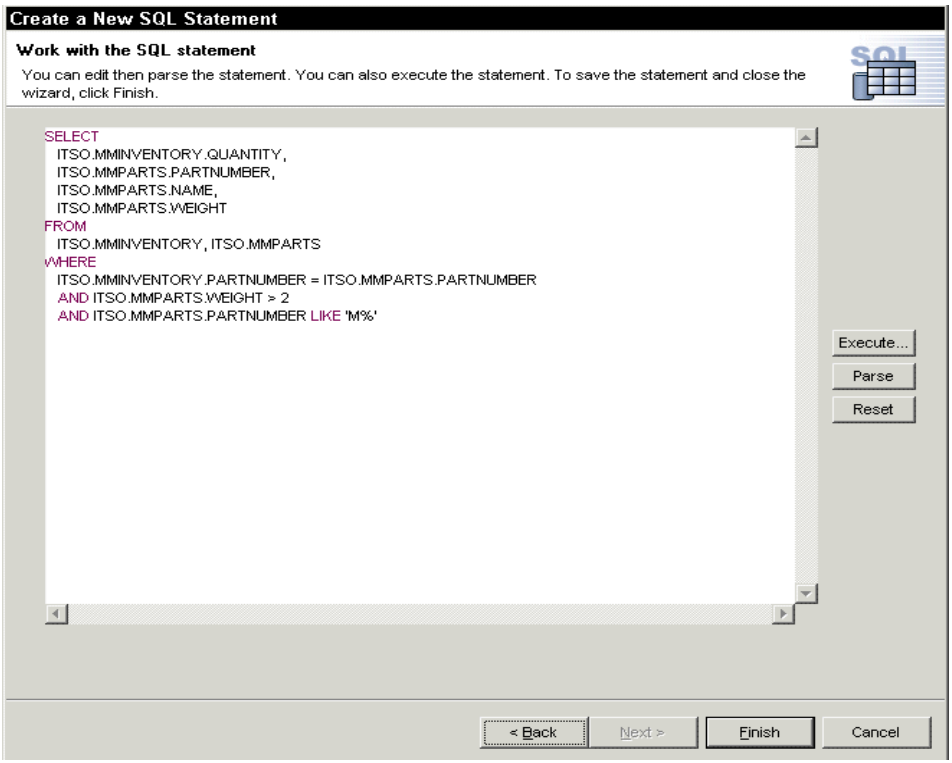


Figure 10-6 SQL Statement Wizard - generated SQL statement

If you want, you can edit the statement directly. When you're finished editing, you can click **Parse** to validate that the SQL statement is correct.

To test the SQL statement, you click **Execute** and then **Execute** again on the next window to see the result of the query Figure 10-7.

Restriction: If you do make changes to the statement on this page, you can no longer go back to the previous page to change the visual design of the query. To be able to do this you first need to click Reset to back out any manual changes you have made.

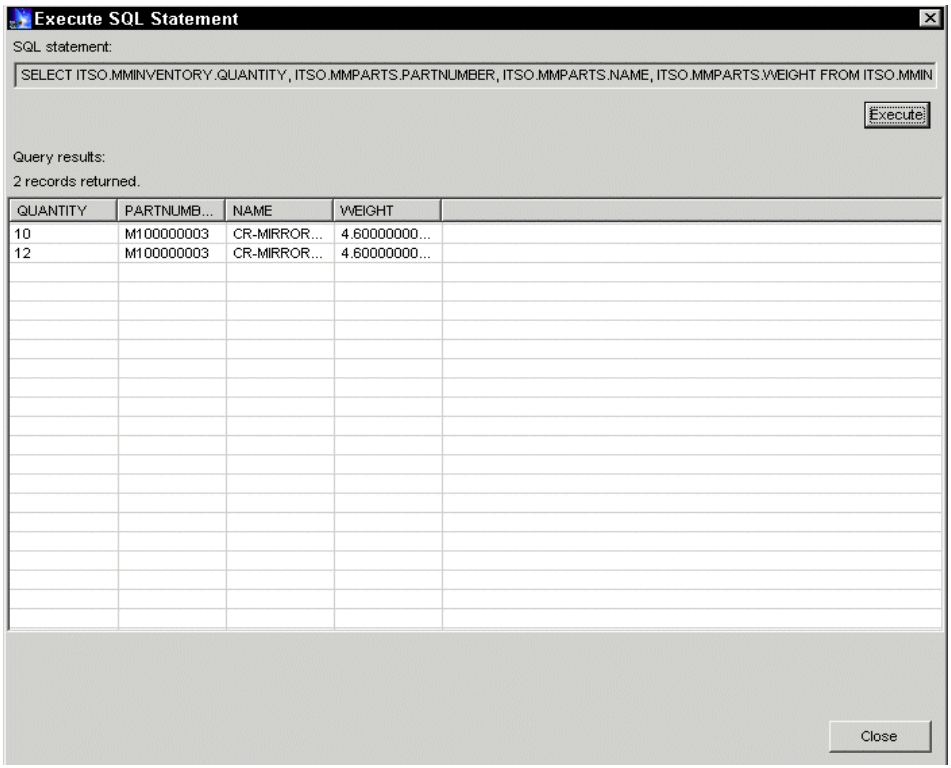


Figure 10-7 SQL Statement Wizard - test SQL statement

10.2 Using SQL Query Builder

The other way of creating SQL statements in Application Developer is to use the *SQL Query Builder*. This tool supports all the options of the SQL wizard, with the addition of `WITH` and `FULLSELECT`. In this section we will describe how to use the SQL Query Builder to build the same `SELECT` statement as we did using the SQL Wizard in the previous section.

To reiterate, for our example we will be developing a `SELECT` statement against the sample `ITSOWSAD` database. We would like to select all parts from the `MMPARTS` table with a part number beginning with "`M`" and that have a weight greater than 2. We would also like to show the quantity on hand for these parts.

To start the *SQL Query Builder*, expand the folder of the database you want to work with until you see the **Statements** folder. Right-click it and select **New—>Select Statement**. A dialog to enter the name of the statement will be displayed. Once you have entered the name, (`ListParts` in the example), and clicked **OK**, the appropriate SQL Builder screen will be displayed (Figure 10-8). To define your query go through the following steps.

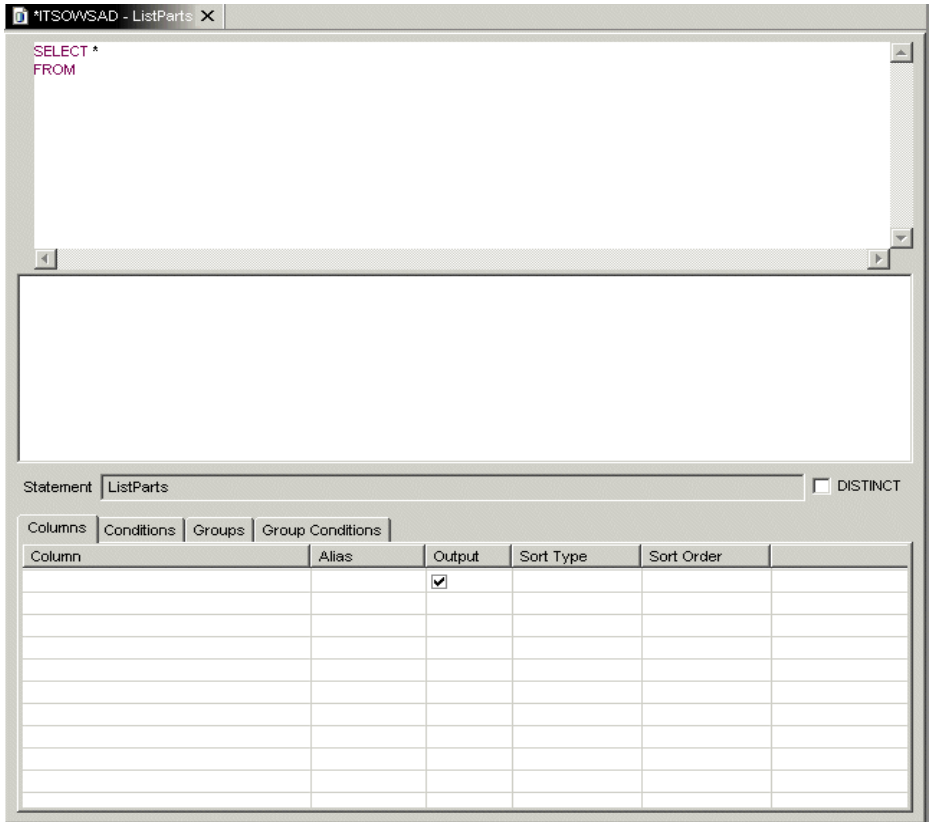


Figure 10-8 SQL Query Builder - create SELECT statement

First we must add the tables that are involved in the query. In our example these are `MMPARTS` and `MMINVENTORY`. To add them simply drag them from the Navigator view and drop them in the middle pane of the SQL Query Builder screen. The result is shown below in Figure 10-9.

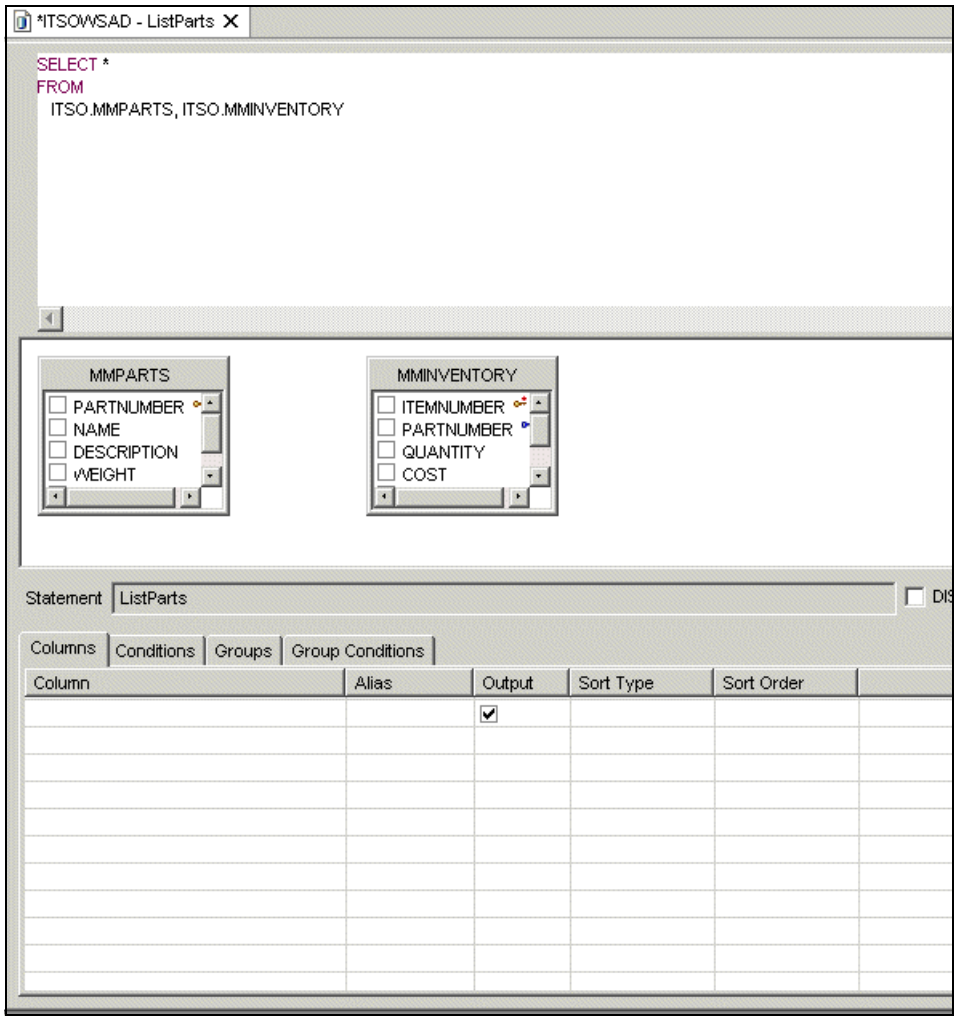


Figure 10-9 SQL Query Builder - adding tables

As you can see the tables have been added to the `SELECT` statement in the top pane.

Next you need to select the columns from each table and join the tables together. To select a column, check the box next to its name. To join the tables, select the `PARTNUMBER` column in the `MMPARTS` table and drag it across to the corresponding column in the `MMINVENTORY` table. A link symbol will be shown between the two and the `SELECT` statement is updated with the corresponding `WHERE` clause (Figure 10-10).

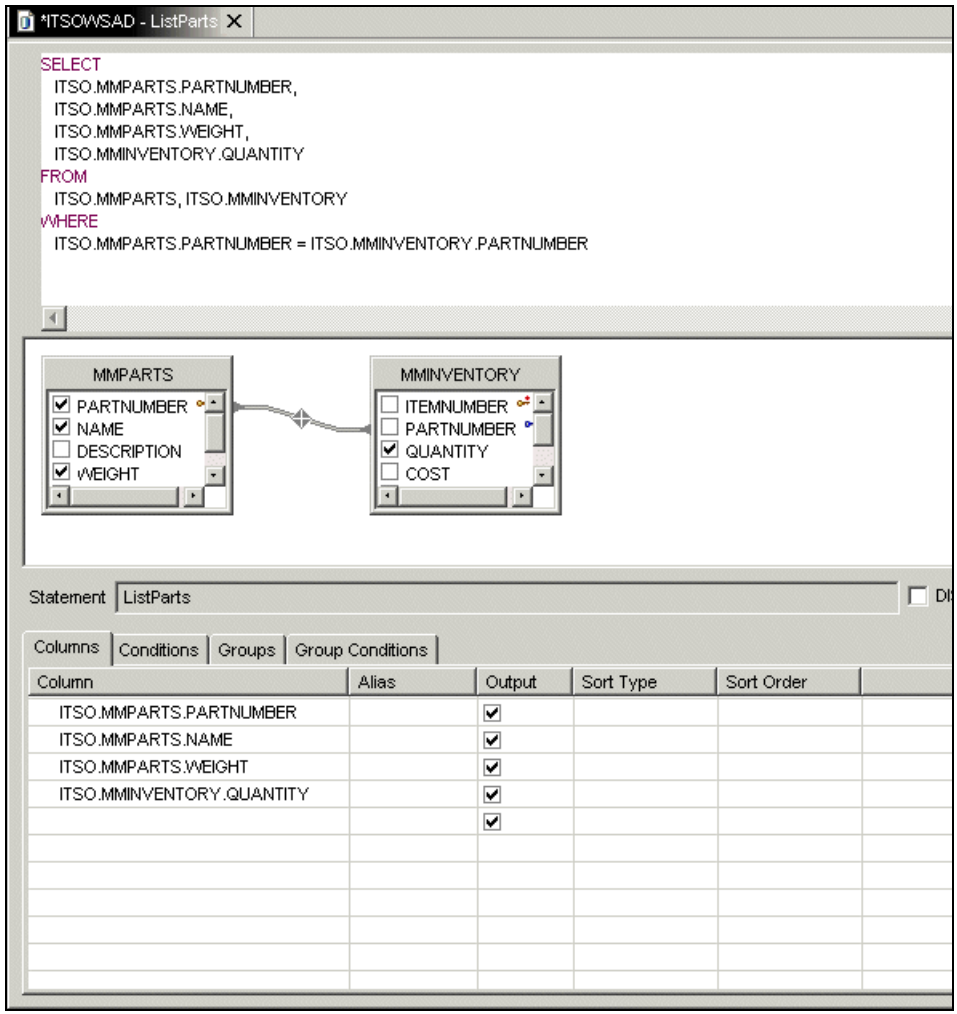


Figure 10-10 SQL Query Builder - selecting columns and joining tables

Finally we want to add the two conditions (`PARTNUMBER LIKE 'M%' and WEIGHT > 2`). Use the *Conditions* tab in the bottom pane to add them. You can also type them directly into the statement and the Conditions tab will be updated. After this has been done the screen will look like Figure 10-11.

Note: Unlike the SQL Statement Wizard, you *can* make manual changes to the generated statement, and have those changes reflected in the visual query builder.

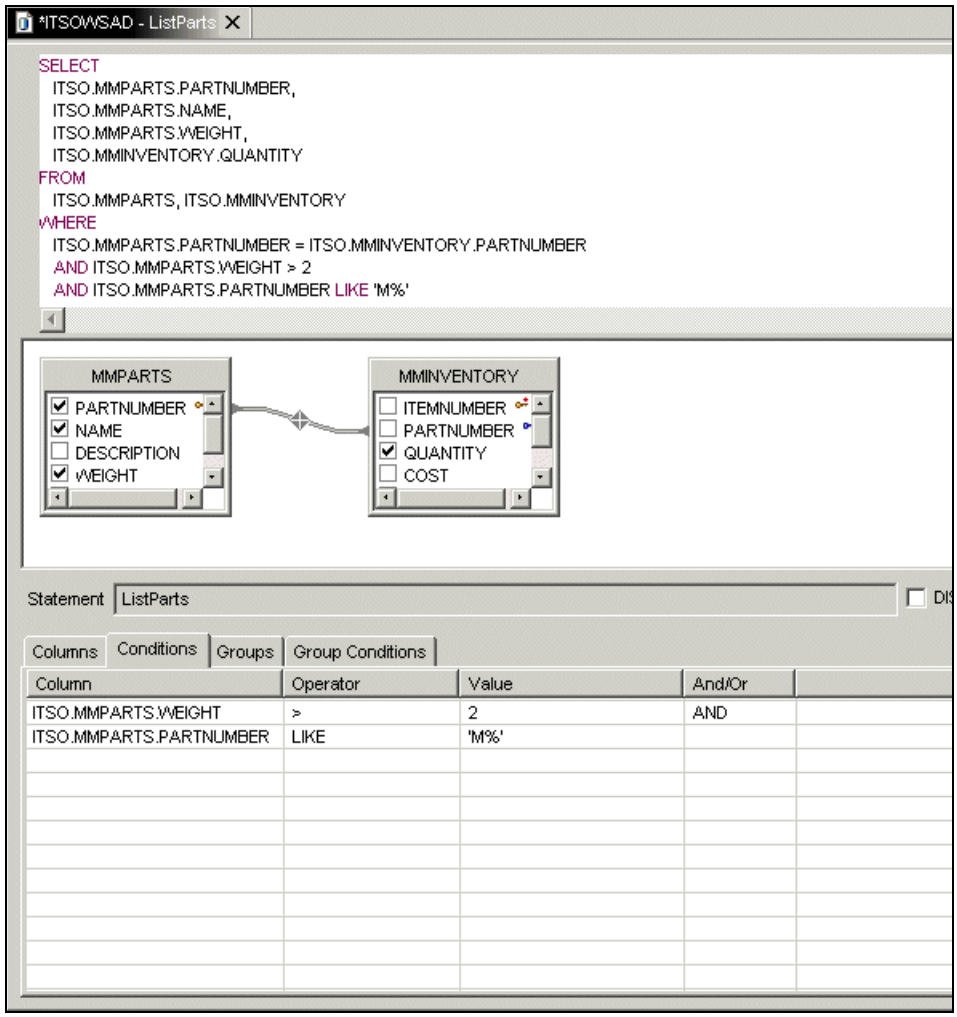


Figure 10-11 SQL Query Builder - adding the conditions

To test the statement, right-click it in the Statements folder and select **Execute**. Click the **Execute** button on the next screen and the matching rows from the database will be shown as in Figure 10-12.

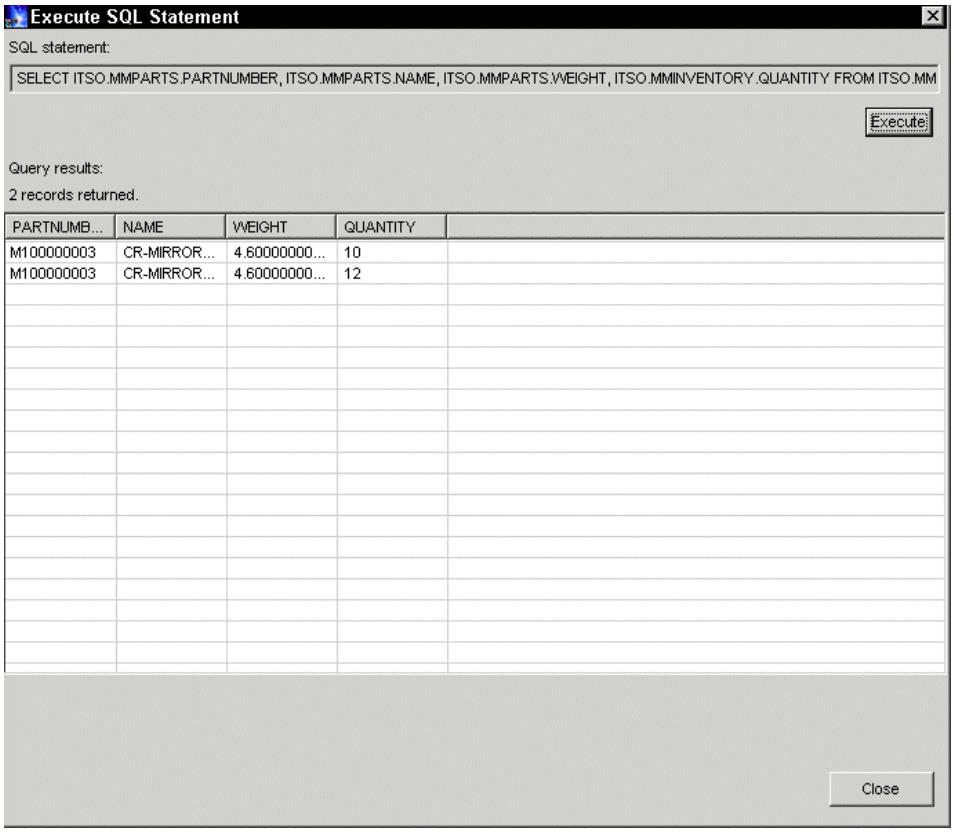


Figure 10-12 SQL Query Builder - testing the statement

Later on we will look at how you can use Application Developer to generate Web pages and Java classes to quickly and easily build an application based on an SQL statement. See “Generate Web pages from SQL queries” on page 304.

11

Stored procedures

This chapter discusses the following topics:

- What is a stored procedure?
- Creating a Java stored procedure
- Accessing a Java stored procedure

© Copyright IBM Corp. 2002

11.1 What is a stored procedure?

Important: The support for stored procedures varies between different DBMSs. We will be using DB2 stored procedures in the example.

A stored procedure is a block of procedural constructs and embedded SQL statements that are stored in a database and can be called by name. Stored procedures allow an application program to be run in two parts, one on the client and the other on the server, so that one client-server call can produce several accesses to the database. This is good for performance since the traffic between the client and the server is minimized.

Stored procedures can be written as SQL procedures or as C or Java programs. In the next sections we will look at how to write and use a Java stored procedure.

11.2 Creating a Java stored procedure

Stored procedures are created outside of Application Developer. A graphical Stored Procedure Builder is included in DB2 UDB to make it easier to create a Java stored procedure. This tool can be accessed from the Windows Start menu by selecting **Programs—>IBM DB2—>Stored Procedure Builder**. For more information about this tool, please see the DB2 documentation. If you prefer, you can of course choose to manually create the procedure. Once the procedure is created it is loaded into the database and is then available for use by a client application. If you use the Stored Procedure Builder, the loading into the database can be done automatically for you.

Below is an example of a DB2 command required to manually load a Java stored procedure into the database:

```
CREATE PROCEDURE PartListing ( IN partnum char(10) ) EXTERNAL NAME
'itso.wsad.dealer.parts.sp.PartListing.partListing' RESULT SETS 1 LANGUAGE JAVA
PARAMETER STYLE JAVA NOT DETERMINISTIC FENCED NO DBINFO NULL CALL MODIFIES SQL
DATA
```

This creates a stored procedure called `PartListing` which takes one argument, (`partnum`). The implementing Java class is in a package called `itso.wsad.dealer.parts.sp` and is named `PartListing`. The method that will be called is `partListing`.

The code for the sample procedure is shown below:

```
/**
 * JDBC Stored Procedure PartListing
```

```
 */
package itso.wsad.dealer.parts.sp;

import java.sql.*;                    // JDBC classes

public class PartListing
{
    public static void partListing ( String partnum,
                                     ResultSet[] rs ) throws SQLException,
Exception
    {
        // Get connection to the database
        Connection con = DriverManager.getConnection("jdbc:db2:itsowsad");
        PreparedStatement stmt = null;
        String sql;

        sql = "SELECT"
            + "    ITSO.AAPARTS.NAME AS NAME,"
            + "    ITSO.AAPARTS.DESCRIPTION AS DESCRIPTION,"
            + "    ITSO.AAPARTS.WEIGHT AS WEIGHT"
            + " FROM"
            + "    ITSO.AAPARTS"
            + " WHERE"
            + "    ("
            + "      ( ITSO.AAPARTS.PARTNUMBER like  ?  CONCAT '%' )"
            + "    )";
        stmt = con.prepareStatement( sql );
        stmt.setString( 1, partnum );
        rs[0] = stmt.executeQuery();
        if (con != null) con.close();
    }
}
```

This simple procedure takes an input argument (a partial part number) and returns the name, description, and weight of any matching parts in the database.

11.3 Accessing a Java stored procedure

Once you have loaded a stored procedure into the database, it can be used from within Application Developer. The most straightforward way is to create a JavaBean that calls the stored procedure. The code fragment below shows an example of how to call the stored procedure:

```
CallableStatement cs = con.prepareCall("{call PartListing(partNum)} ");
ResultSet rs = cs.executeQuery();
while (rs.next()) {
   // get the data from the row
```

```
}
```

The other way of accessing the stored procedure is from a JSP using either JSP tags or the DB Beans library. These methods will be described in “Accessing databases from a Web application” on page 303.

You can use the “Create Web pages from a JavaBean” wizard to build the input and output pages to test the JavaBean that calls the stored procedure. This wizard is described in “Creating Web pages from a JavaBean” on page 182.

You can find code listings for the sample Java stored procedure (PartListing.java) and the JavaBean calling it (`PartListBeanSP.java`) in Appendix C, “Additional material” on page 675.

12

Accessing databases from your applications

This chapter discusses the following topics:

- Accessing databases from a Java application
- Accessing databases from a Web application
- Generating Web pages from an SQL query
- Accessing databases using DB Beans
- Accessing databases using JSP tags

© Copyright IBM Corp. 2002

12.1 Accessing databases from a Java application

We have already seen an example of how to access a relational database via JDBC from a Java application. In “Java applications” on page 140 we created a simple application that accessed a DB2 table to return a list of parts.

We will now take a closer look at how the database access is done. We use the driver manager class to manage the connection to the database. As has been discussed earlier, “Data source versus direct connection” on page 263, JDBC 2.0 has an alternative way of handling connections using connection pooling and Data sources.

First we need to establish a connection to the database. The following code fragment shows how to do this:

```
protected static Connection connect() {
   Connection con = null;
   try {
   Class.forName("COM.ibm.db2.jdbc.app.DB2Driver")
   con = DriverManager.getConnection("jdbc:db2:itsowsad");
    catch(Exception e) {...}
      return con;
```

The first thing that needs to be done is to load the JDBC driver. The `Class.forName()` call does this. The driver name is dependent on which database you are connecting to.

DB2 supplies two different JDBC drivers:

- **COM.ibm.db2.jdbc.app.DB2Driver:** This is a JDBC Type 2 driver that uses a DB2 client installed on the machine where the application runs. You would use this driver when accessing a local database or a remote database via a local DB2 client.
- **COM.ibm.db2.jdbc.net.DB2Driver**: This is a JDBC Type 3 driver. It’s a pure Java driver that is designed to enable Java applets access to DB2 data sources. Using this driver your application will talk to another machine where the DB2 client is installed.

In our examples we use the **app** driver since we are talking to a local database.

Note: There is no need to create an instance of the driver or register it. This is done automatically for you by the DriverManager class.

After loading the driver, you need to establish a connection. The class that handles this is called DriverManager. The URL string that is passed in to the `getConnection()` method is again dependent on which database system you're using. In the example above we are connecting to a DB2 database called itsowsad. In this example we are not passing a userid and password, but if that was required, they would be the second and third parameters of the getConnection()call.

If you are trying to connect to another database system you need to consult the documentation to determine what driver name and URL to use.

The classes that you need to have access to when connecting to a DB2 database from Java are found in **.\sqllib\java\db2java.zip**. You would make this available in Application Developer by creating a class path variable for it and adding that to the project build path.

You are now ready to perform operations on the database. For a simple select, the code would look like this:

```
stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from itso.aaparts");
```

You create a statement using the connection obtained from the DriverManager and then you execute the query passing the select statement. The result set from the query is returned in a ResultSet variable.

Finally you need to process the result set from the query. The ResultSet class provides a number of get...() methods for various data types that can be used for this.

```
while (rs.next()) {
   String partnum = rs.getString("partNumber");
   double weight  = rs.getDouble("weight");
   String url     = rs.getString("image_url");
}
```

12.2 Accessing databases from a Web application

There are a number of ways that you can access databases from a Web application. You can of course write your own Java classes and access the database via standard JDBC calls. As an alternative, Application Developer supplies a library of database access beans called DB Beans. These can be used in a JSP through the useBean action and can also be accessed through a

set of JSP tags supplied in a tag library. An application using DB Beans or tags can be generated for you by Application Developer using a wizard based on an SQL statement. See “Generate Web pages from SQL queries” on page 304 for details on the wizard.

Which of these methods you choose will depend on the nature of your application and the complexity of your database access. From the perspective of separating the layers of your application, using separate JavaBeans to do the database access may be more appropriate since you are not mixing presentation and database logic in the JSP as you do when using DB Beans or JSP tags.

In the following sections we will discuss the wizard that can be used to create a View bean or a Taglib application starting from an SQL statement. We will then look a bit more closely at the DB Beans classes and JSP tags.

12.2.1 Generate Web pages from SQL queries

Application Developer provides a wizard to help you create a set of Web pages and supporting Java classes starting from an existing or new SQL query. The wizard will generate the required HTML pages, JSPs and Java classes to quickly create a working skeleton application without you having to write any code. You can then expand and modify the generated code to create the finished application.

To start the wizard, click the database wizard icon  from the toolbar.

The first page of the wizard will be displayed (Figure 12-1).

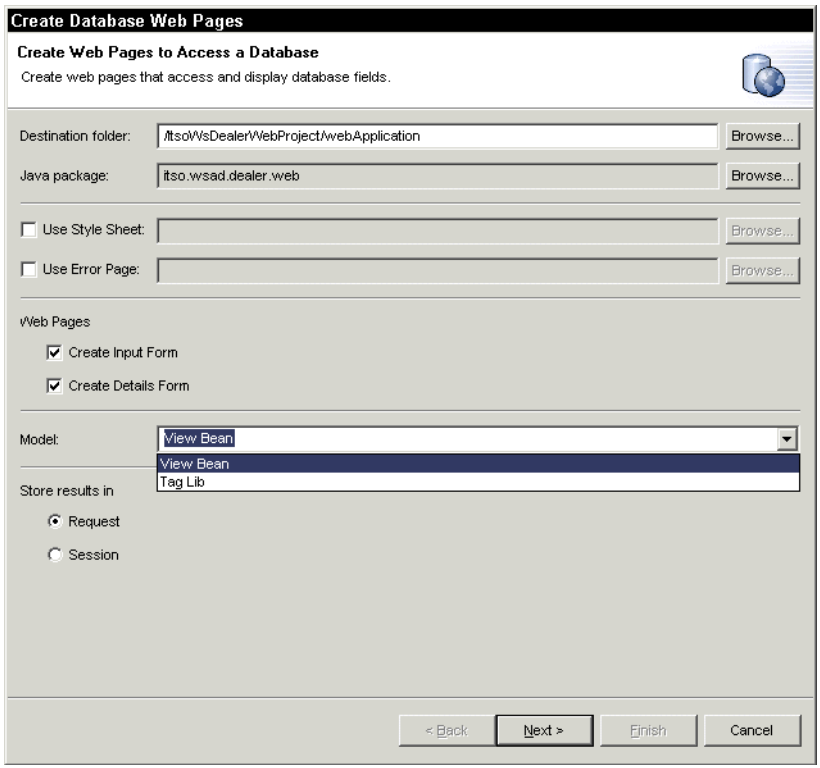


Figure 12-1 Create Database Web Pages wizard - page 1

The following fields are available on this page:

- **Destination folder:** Folder to store the generated Web pages in.
- **Java package:** Package to store the generated Java code in.
- **Use Style Sheet:** If you have a Style Sheet, (CSS file), that you want the wizard to apply to the generated pages, you can specify it here.
- **Use Error Page:** This is the URL of a page that you want displayed in case of an error when running the generated application. If no error page is specified, a default error page will be displayed.
- **Create Input Form:** This option causes the wizard to create a form where you can enter parameters that need to be passed to the SQL statement.
- **Create Details Form:** The wizard will by default generate a list form to display there results of a query. You can optionally also have it create a Details Form that will display the details for each row when selected.
- **Model:** You can have the wizard generate either a View Bean or a Taglib application. The available models are similar; however, the View Helper

nodes vary. In the case of the View Bean model, the view helpers are Java-wrapper classes that manage all database interaction. In the case of the Tag Lib model, a set of custom JSP tags are used to manage database interactions. JSP tags will be discussed in more detail in “Accessing a database using JSP taglib” on page 316. For this example we will be using the View Bean model**.**

- **Request/Session:** This option determines where the results from the query should be stored. You can choose to store them in the session, in which case they will be available to other pages for the life of the session, or in the request. You should be aware of potential memory issues if you choose to store a large result set in the session.

Clicking **Next** will bring up the second wizard page (Figure 12-2).

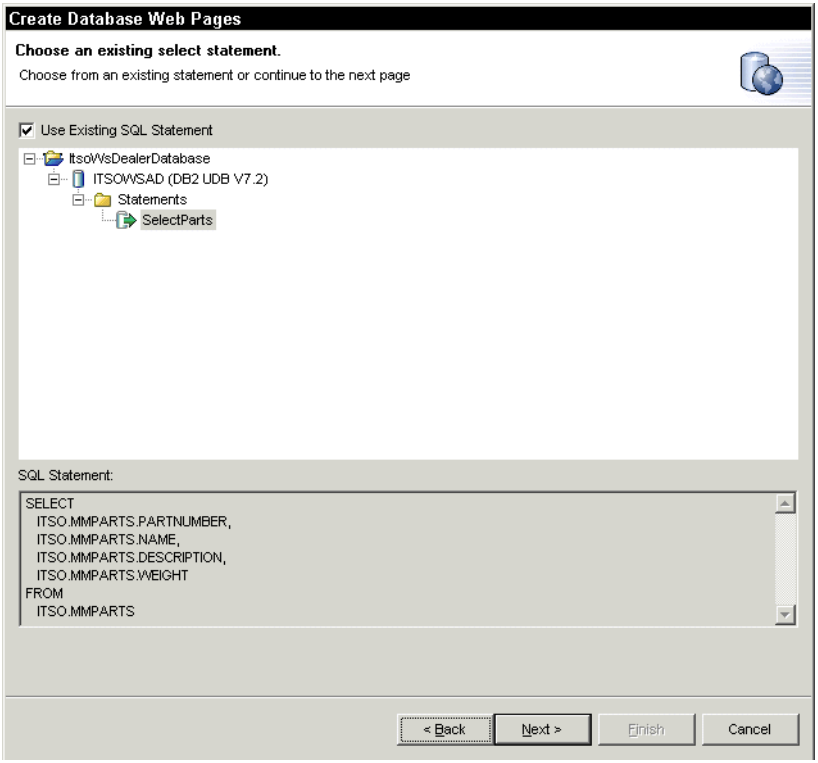


Figure 12-2 Create Database Web Pages wizard - select SQL statement

If you already have an existing SQL statement in the statement folder of a database project, you can select it here and use that to generate the Web pages. For our example we will leave the **Use Existing SQL Statement** check box un-selected and click **Next** to build the statement from scratch using the wizard (Figure 12-3).

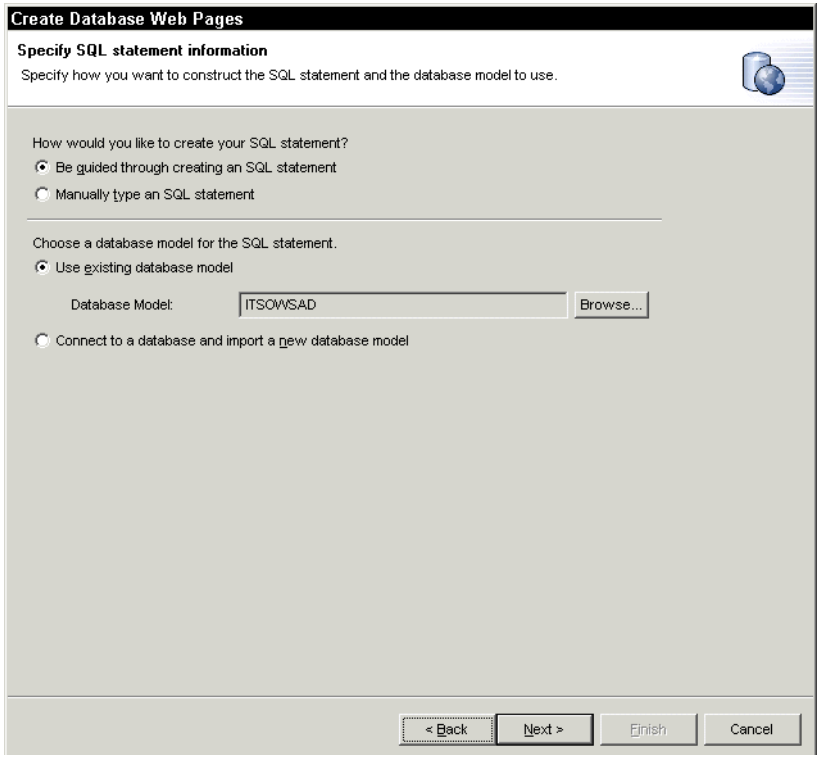

Create Database Web Pages
Specify SQL statement information
Specify how you want to construct the SQL statement and the database model to use.
How would you like to create your SQL statement?
Be guided through creating an SQL statement
Manually type an SQL statement
Choose a database model for the SQL statement.
Use existing database model
Database Model:
ITSOWSAD
Browse...
Connect to a database and import a new database model
< Back
Next >
Finish
Cancel

Figure 12-3 Create Database Web Pages wizard - statement options

If you want, you can elect to type the SQL statement in manually. If you want more help, select the guided option. If you have a database model created already you can choose to use it now. Alternatively you can connect to the database and import a model. See Chapter 9, “Database connectivity” on page 261 for more details on database models and how to create them.

In this case we are assuming that we already have created a data model for the database ITSOWSAD and that we will use the wizard facilities for building the SQL statement. Clicking **Next** will display the page for building the SQL statement (Figure 12-4).

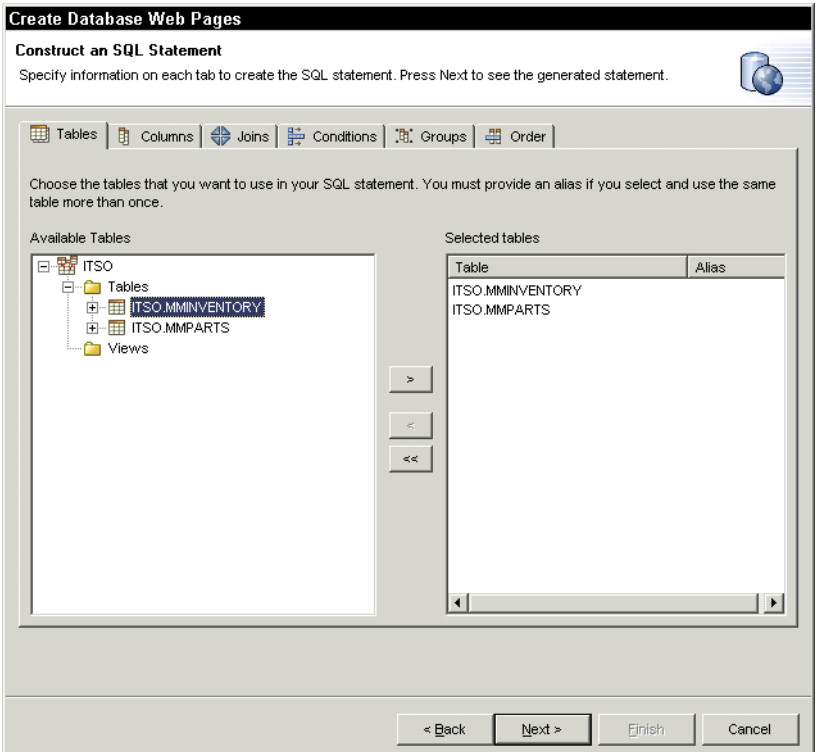


Figure 12-4 Create Database Web Pages wizard - build SQL statement

How to build the SQL statement using the wizard pages has been described earlier in the discussion in “Using the SQL Wizard” on page 284, so we won’t go into the details of the various tabs here. Once you have defined the tables, columns, joins, conditions and grouping and ordering of the results, you click **Next** to view the generated statement. Figure 12-5 shows the finished statement for our example.

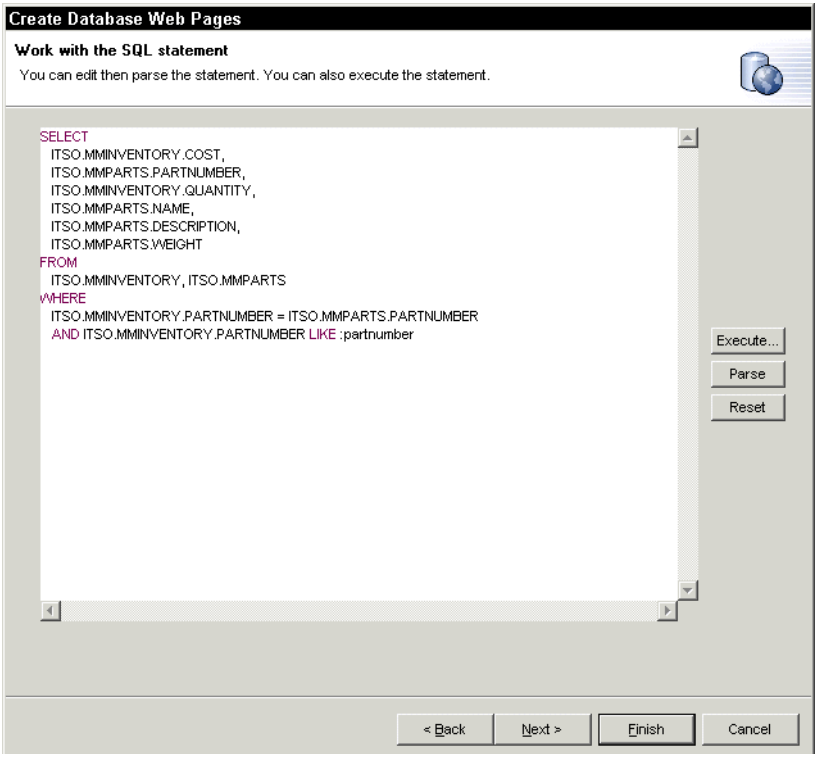


Figure 12-5 Create Database Web Pages wizard - finished SQL statement

Clicking **Next** will display the page where you decide how you want to access the database when the generated application is run (Figure 12-6). You can choose to use a direct connection or use a Data source. For a discussion about these two different ways of connecting, see “Data source versus direct connection” on page 263.

In this case we will to use the driver manager connection. Normally the fields will be populated with the driver name and database URL.

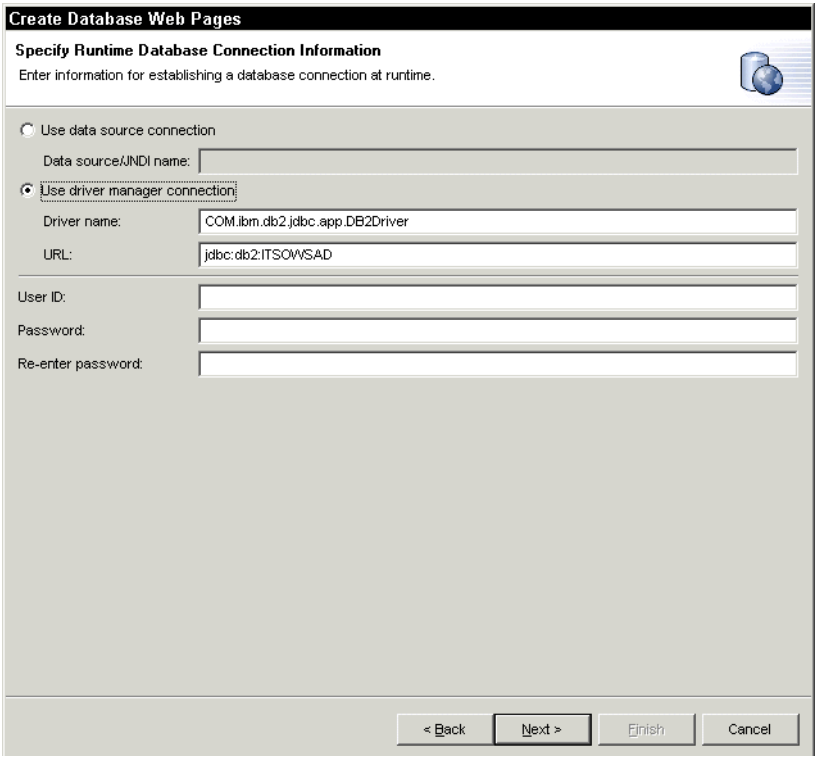


Figure 12-6 Create Database Web Pages wizard - connection information

On the next three wizard pages you can view and change the pages that will be generated by the wizard. On the first page you will see the HTML input form (Figure 12-7).

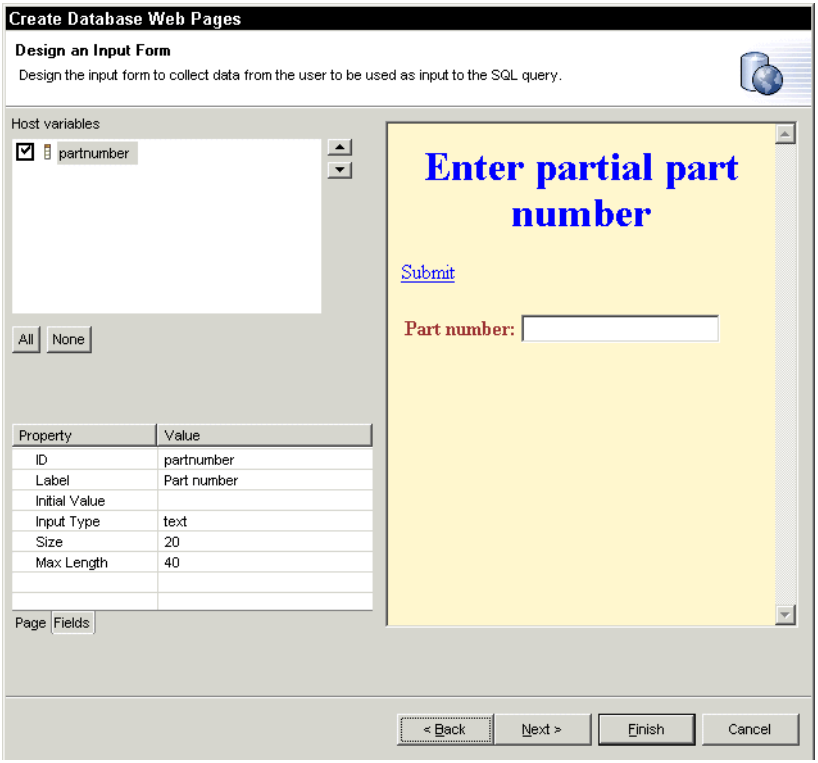


Figure 12-7 Create Database Web Pages wizard - design input form

Here you can make some changes to page and field properties. As with the following pages, once they have been generated you can make further changes as with any other page.

The next page shows the master result page. The default is to use a table to display the result rows from the query (Figure 12-8).

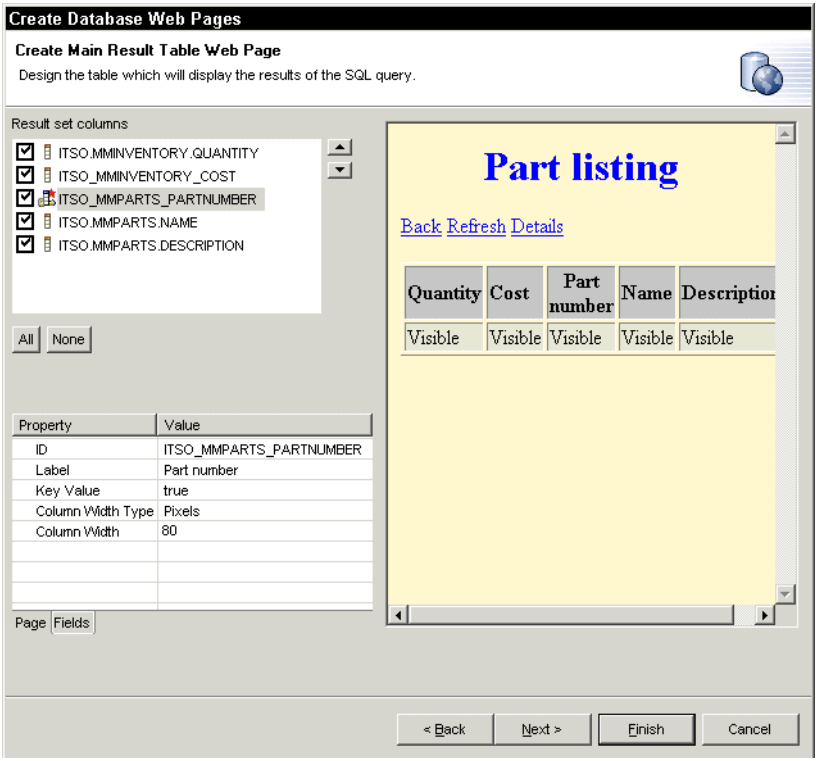


Figure 12-8 Create Database Web Pages wizard - design result form

Here you can change the heading of the columns and page properties. If you don't want to show one or more of the fields retrieved from the query you can deselect them in the top left pane. To change the labels, select the item in the top left pane and make your changes in the bottom left pane.

Note: Be aware that the changes will only take effect when you click on another property or value in the pane.

The next page shows the default form for showing details of a selected row in the Result page (Figure 12-9).

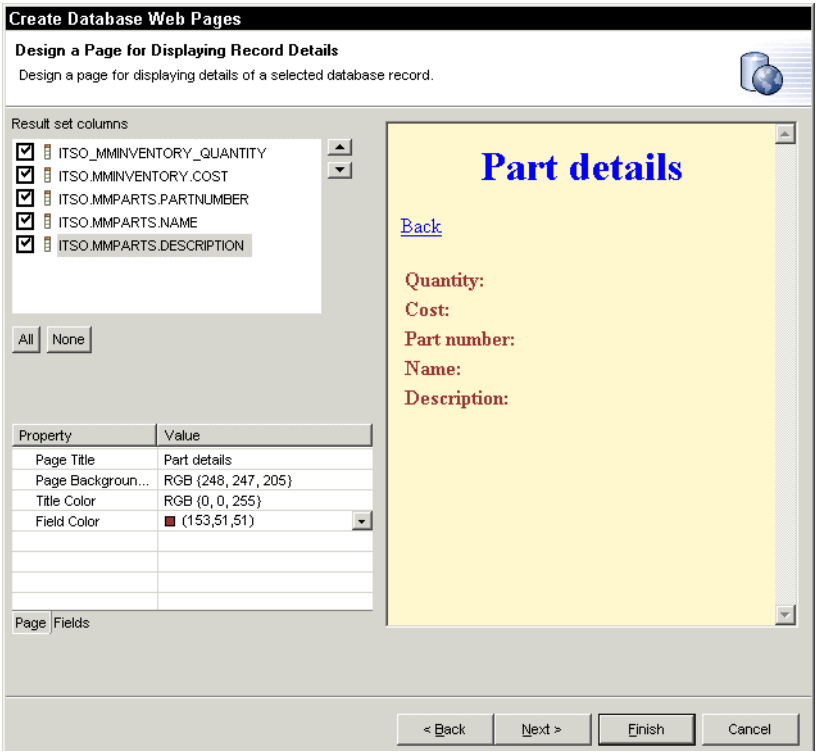


Figure 12-9 Create Database Web Pages wizard - design detail form

You can make the same type of changes here as on the other pages to improve the look of the page.

On the final two pages you can make some more choices about the generated objects. First you can specify whether you want to create a new controller servlet, use an existing one or not use one at all (Figure 12-10).

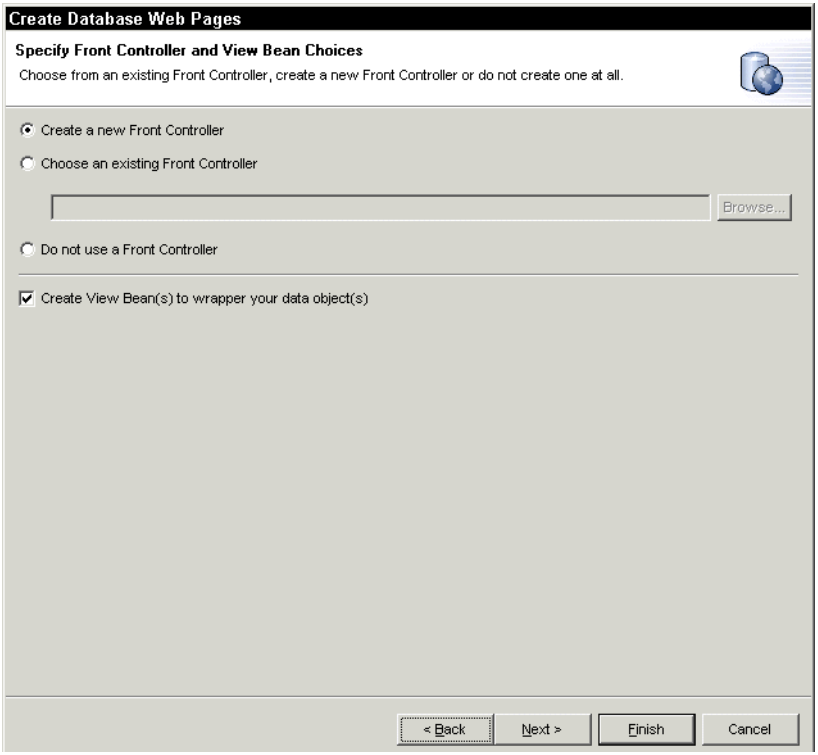

Create Database Web Pages
Specify Front Controller and View Bean Choices
Choose from an existing Front Controller, create a new Front Controller or do not create one at all.
Create a new Front Controller
Choose an existing Front Controller
Browse...
Do not use a Front Controller
Create View Bean(s) to wrapper your data object(s)
< Back
Next >
Finish
Cancel

Figure 12-10 Create database Web pages wizard - select controller

On this page you can also tell the wizard if you want it to create the View Bean wrapper classes or not. If you deselect this option, the wizard will access the query result using the underlying connection beans directly.

Finally you can modify the default prefix that will be used for the generated objects (Figure 12-11).

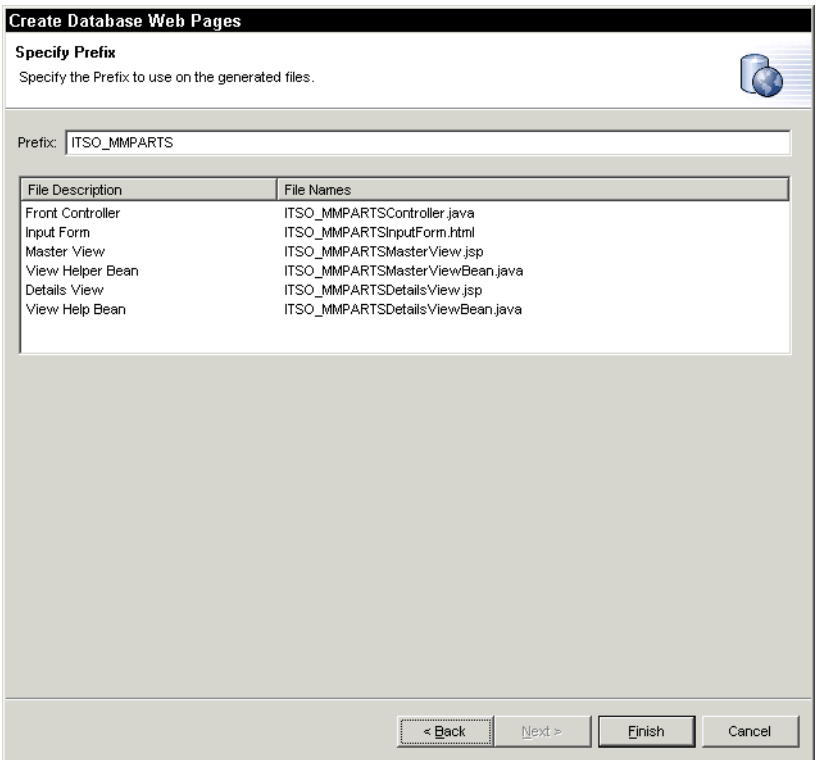


Figure 12-11 Create database Web pages wizard - select prefix

Clicking **Finish** will generate the Java classes and HTML/JSP pages. To test the generated application, select the generated HTML input form, (`ITSO_MMPARTSInputForm.html` in this example), and select **Run on Server** from its context menu.

12.2.2 Accessing a database using DB Beans

The DB Beans classes can be found in the com.ibm.db.beans package. To access them from Application Developer you need to import the following jar file:

`..\plugsins\com.ibm.etools.webtools.jars\`*`dbbeans.jar`*

The documentation for the classes in the package can be found in:

`..\plugins\com.ibm.etools.webtools\jars\`*`dbbeans_javadoc.zip`*

Restriction: Currently there is no online documentation for this package in the Application Developer Help Perspective.

After you have imported the package you can use the DB Beans classes by using jsp:useBean to create the bean and then using scriptlets to execute methods on it. Here is simple example of a JSP that executes an SQL statement using the DB Beans classes:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>TestDBBeans.jsp</TITLE>
</HEAD>
<BODY>
<H2>Number of rows in AAPARTS table</H2>
<jsp:useBean id="Connect" class="com.ibm.db.beans.DBConnectionSpec"/>
<jsp:useBean id="SelectStatement" class="com.ibm.db.beans.DBSelect"/>
<% Connect.setDriverName("COM.ibm.db2.jdbc.app.DB2Driver");
Connect.setUrl("jdbc:db2:itsowsad");
SelectStatement.setConnectionSpec(Connect);
SelectStatement.setCommand("SELECT * FROM ITSO.AAINVENTORY");
SelectStatement.execute();
out.println("Rowcount is: " + SelectStatement.getRowCount());

%>
</BODY>
</HTML>
```

Note: This code was created using Page Designer by visually inserting beans and scriptlets. To test the JSP, select **Run on Server** from its context menu.

In this example we use two of the DB Beans classes: DBConnectionSpec, which handles the database connection, and DBSelect, which wraps an SQL SELECT statement.

There is a small number of classes in the DB Beans package, and they are straightforward to use. The next section describes the JSP tags that have been built on top of the beans to make it even easier to provide database access functionality to your Web application.

12.2.3 Accessing a database using JSP taglib

Application Developer provides an alternative to using the DB Beans classes described above. If you prefer, you can instead use a set of JSP tags built on top of these classes. It is also possible to mix direct calls to DB Beans and JSP tags.

If you decide to use the JSP tags you should be aware that there are some restrictions compared to using the beans directly:

- For any of the JSP SQL actions that require a connection to the database, a connection will be opened when the tag is encountered, and closed after the tag has been processed. Two actions cannot be performed within the same transaction scope, or even using the same JDBC connection. The only exception to this is via the x:batch action. Actions inside the body of the x:batch do share the same connection, and optionally, the same transaction.
- Using the DB Beans directly, you have complete control over when a database connection is opened and closed. You also have complete control over transaction scopes, with the ability to turn *AutoCommit* on or off and to do explicit commits or rollbacks.
- Some of the methods and properties of the DBSelect and DBProcedureCall beans for handling large result sets are not offered via the JSP SQL actions. These methods and properties allow you to limit the number of rows maintained in memory at any one time and to specify how many rows to fetch at once when getting additional rows. This limitation is necessary because of the above limitation that the database connection is closed after each JSP SQL action is processed. If only a subset of the rows is initially fetched into memory, and then the connection is closed, there is no way to later fetch the remaining rows. The JSP SQL actions do provide some support for large result sets via the maxRows attribute of the x:select and x:procedureCall actions. This attribute simply limits the number of rows that will be fetched in any one result set. The lockRows property of the DBSelect and DBProcedureCall bean is not offered via the JSP SQL actions. This property causes a database lock to be kept on a row in the result set while it is the current row. For a Web application, it is not likely that you would want to maintain such a lock across user interactions which could span an arbitrary amount of time. Because of the first limitation above, that the database connection is closed after each JSP SQL action is processed, it is not possible for us to maintain such a lock when you use the JSP SQL tags. When row locking is not used, either with the JSP SQL tags or with direct use of the DB Beans, "optimistic" locking is still used to prevent you from updating a row if someone else updates it between the time that you read it and the time that you attempt to update it. A greater variety of methods for moving between rows and between result sets is available through direct use of the DB Beans than through the JSP SQL actions.

To use the JSP database tags you need to import the following two JAR files into the WEB-INF\lib folder of your project:

```
...\plugsins\com.ibm.etools.webtools.jars\jspsql.jar
...\plugsins\com.ibm.etools.webtools.jars\dbbeans.jar
```

The *jspsql.jar* contains the tags and *dbbeans.jar* contains the actual classes used to access the database.

You can use the JSP editor to insert the database tags into your page when you are in the Design view. To be able to use the custom tags from the database tag library, you first need to do two things:

1. Import the tag library into your Web application as described above.
2. Create the taglib directive in the JSP.

To insert the taglib directive, bring up the context menu on the page in the Design view and select **Page Properties**. In the dialog shown select the **JSP Tags** tab and **JSP Directive - taglib** from the Tag Type drop-down menu (Figure 12-12).

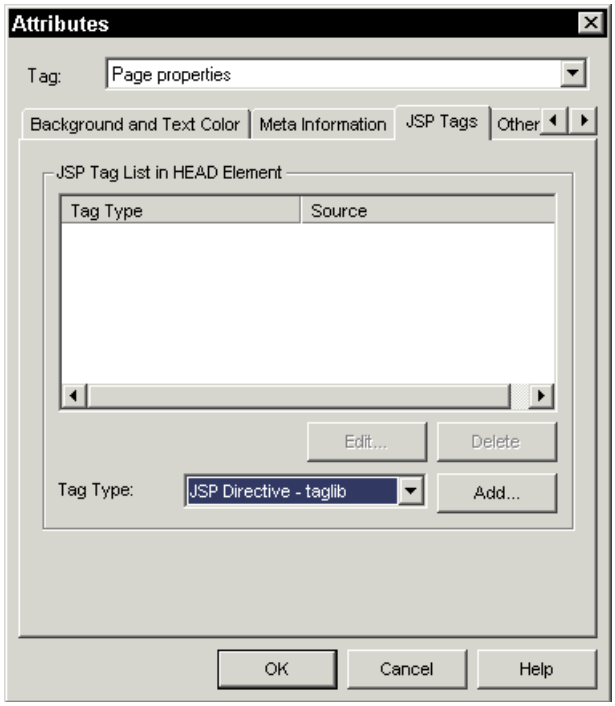


Figure 12-12 Insert JSP taglib directive

Click **Add** to select the tag library to add (Figure 12-13). Locate the jspsql.jar file and then enter the **Tag Prefix** that will be used. The prefix can be whatever you like. Whenever you use one of the tags from the taglib, it must be prefixed with the string specified in the taglib directive.

Click **OK** and **OK** again to update the page properties. The following tag will be inserted into your JSP:

```
<%@ taglib uri="/WEB-INF/lib/jspsql.jar" prefix="dab" %>
```

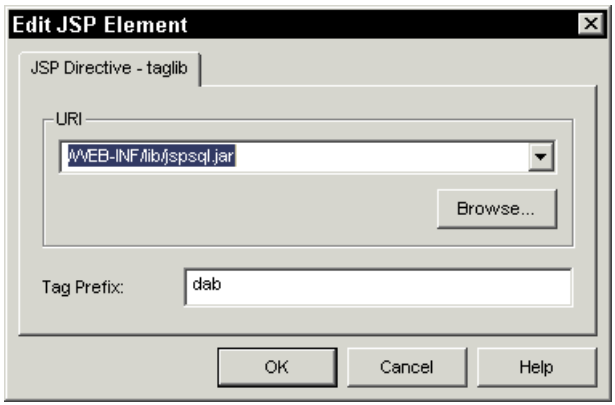


Figure 12-13 Selecting the tag library and prefix

Now you can add tags to the JSP by selecting **JSP—>Insert Custom**. You should see the following dialog (Figure 12-14).

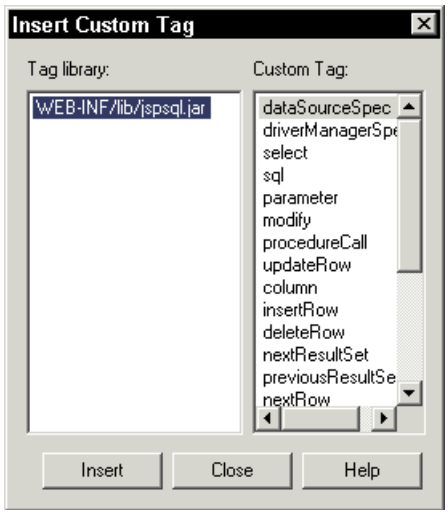


Figure 12-14 Insert custom JSP tags

In the following example we are creating a JSP that will display a list of parts. We will show you some of the main tags that you would use to access the database. The Application Developer online help contains descriptions for all the tags that are available.

Tip: If you want to use the JSP tags, you should consider using the Create Database Web Pages wizard to do the initial work. Even if you don't want to use the rest of the infrastructure, generating the JSP will be quicker than building it using the JSP editor.

First we need to create the connection to the database. To do this we use the *dab:driverManagerSpec* tag.

```
<dab:driverManagerSpec id="Connect" scope="page"
   userid='<%=config.getInitParameter("username")%>'
   password='<%=config.getInitParameter("password")%>'
   driver='<%=config.getInitParameter("driverName")%>'
   url='<%=config.getInitParameter("url")%>'/>
<%
```

Note: All the parameters for the connections are retrieved from the web.xml file containing the deployment information for your project.

Once you have established a connection, you can execute the query:

```
<dab:select id="select_master" scope="request"
connectionSpecRef="Connect>">
   <dab:sql>
      SELECT ITSO.MMINVENTORY.QUANTITY, ITSO.MMINVENTORY.COST,
ITSO.MMPARTS.PARTNUMBER, ITSO.MMPARTS.NAME, ITSO.MMPARTS.DESCRIPTION FROM
ITSO.MMINVENTORY, ITSO.MMPARTS WHERE ITSO.MMINVENTORY.PARTNUMBER =
ITSO.MMPARTS.PARTNUMBER AND ITSO.MMINVENTORY.PARTNUMBER LIKE :partnumber
   </dab:sql>
   <dab:parameter position = "1" type="CHAR" value="<%=inputpartnumber%>"/>
</dab:select>
```

Here you use the connection created previously to issue the SQL select statement. The input parameter is specified using the *dab:parameter* tag. (The variable *inputpartnumber* contains the value of the parameter.)

Assuming that you have created an HTML table to display the result, you can then use *dab:repeat* and *dab:getColumn* to loop through the result set and display the values.

```
<dab:repeat name="select_master" index="rowNum" over="rows" >
   <TR>
      <TD>
         <INPUT TYPE="radio" NAME="selection"
onclick="document.myForm.elements['selected_index'].value=<%=rowNum%>"/>
      </TD>
      <TD>
         <dab:getColumn index="1"/>
```

```
        </TD>
        <TD>
           <dab:getColumn index="2"/>
        </TD>
        <TD>
           <dab:getColumn index="3"/>
           <INPUT TYPE="hidden" NAME="ITSO_MMPARTS_PARTNUMBER<%=rowNum%>"
VALUE='<dab:getColumn index="3"/>'/>
        </TD>
        <TD>
           <dab:getColumn index="4"/>
        </TD>
        <TD>
           <dab:getColumn index="5"/>
        </TD>
     </TR>
     <%select_master.next();%>
     </dab:repeat>
```

As you can see from this discussion above, both DB Beans and the corresponding JSP tags give you an easy and quick way to access relational data directly from a JSP. As was mentioned earlier, you need to be aware of the potential problems of combining presentation and business logic in one JSP. From a Model-View-Controller perspective, a JSP should ideally only implement the presentation layer of your application, while the database access should be handled by JavaBeans. However, if your application is small, or if you are building a prototype, using the database access facilities described here may be a good solution.

Part 4

Migrating your applications

Part 4 describes how to migrate your previously developed applications into WebSphere Studio Application Developer.

© Copyright IBM Corp. 2002

13

Migrating your Java classes

This chapter describes how to use or migrate previously developed Java classes. The following topics are discussed:

- Importing Java classes
- Installing an EAR
- Importing from a WAR
- Importing an EJB jar

© Copyright IBM Corp. 2002

13.1 Importing Java classes

If you need to use classes in external JAR files, you can add the JAR file as a Application Developer class path variable. However, if you need to edit previously developed Java code, you have to import the code into the Application Developer project structure. The steps to create a Java project and import Java code are described in 6.1, “Java applications” on page 140.

13.2 Installing an EAR

EAR is an archive that contains an enterprise application. You can import an EAR into your workbench using the import wizard. Select **File—>Import** menu and then select **EAR file** (Figure 13-1).

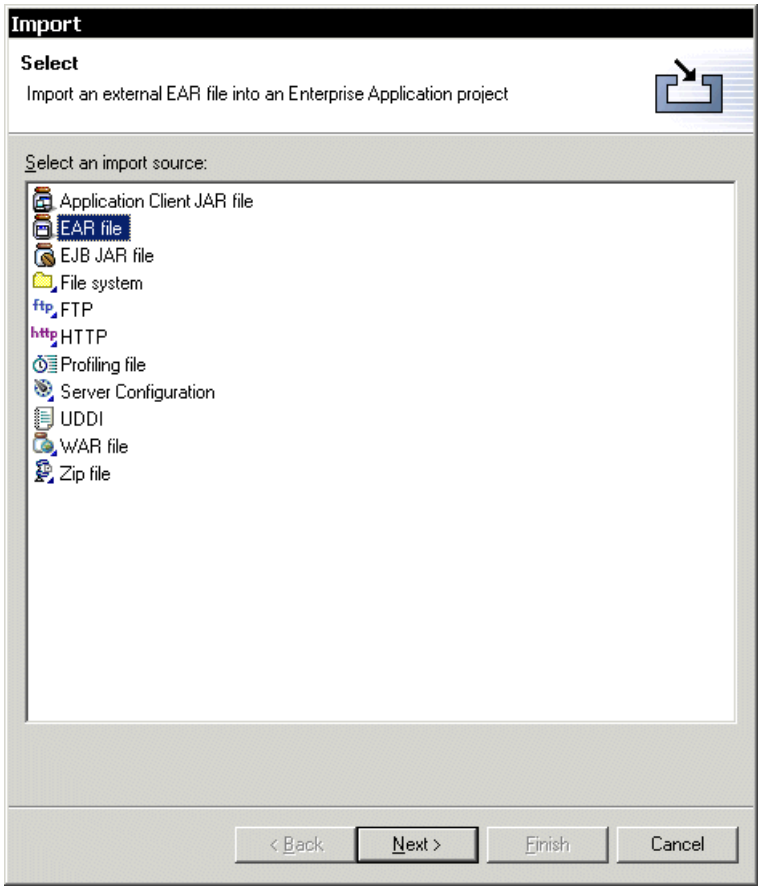


Figure 13-1 Importing an EAR file

Click **Next**, select an EAR file from the file system and name the EAR project into which the file should be imported. In this case, we use the PDKLite package to demonstrate the migration scenario. To get the code for this package follow this link:

http://www.ibm.com/developerworks/patterns/install/index-lite.html

What is PDKLite?: The Patterns for e-business Development Kit Lite (PDK Lite) is a complete, self-configuring, end-to-end skeleton Web application in one self-extracting package. PDK Lite is a "best practice" implementation of User-to-Business topology 1. It contains full source code, WebSphere Studio project, Visual Age for Java project, and NT configuration scripts. The PDK Lite Web application uses servlets, Command beans, JDBC, IBM Data Access Beans, JSP, View beans, XML, and industry-standard best practices to demonstrate the User-to-Business pattern in action

Command_Servers.ear is located in the TestDrive folder. Select the EAR and name the EAR Project to **PDK Command Servers** (Figure 13-2).

Note that Command Server is just part of the whole PDK package. We decided to migrate a smaller piece first to show the migration process.

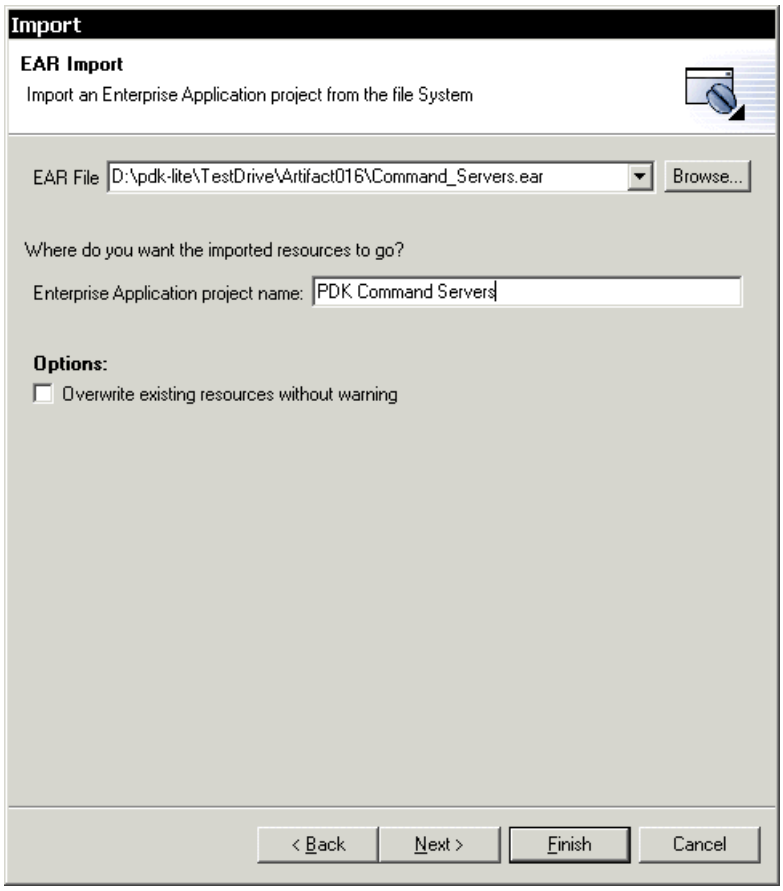


Figure 13-2 EAR import

Command_Servers.ear contains one `.jar` file , one `.war` file, and some descriptor files Figure 13-3.



Figure 13-3 Command_Servers.ear contents

Click **Next**. On the next page the wizard shows the project names that will be used for the import. The default names are:

- EJBCommandTarget

- HttpServletCommandServer

To make the names bit more descriptive change them to (Figure 13-4):

- PDK EJBCommandTarget
- PDK HttpServletCommandServer

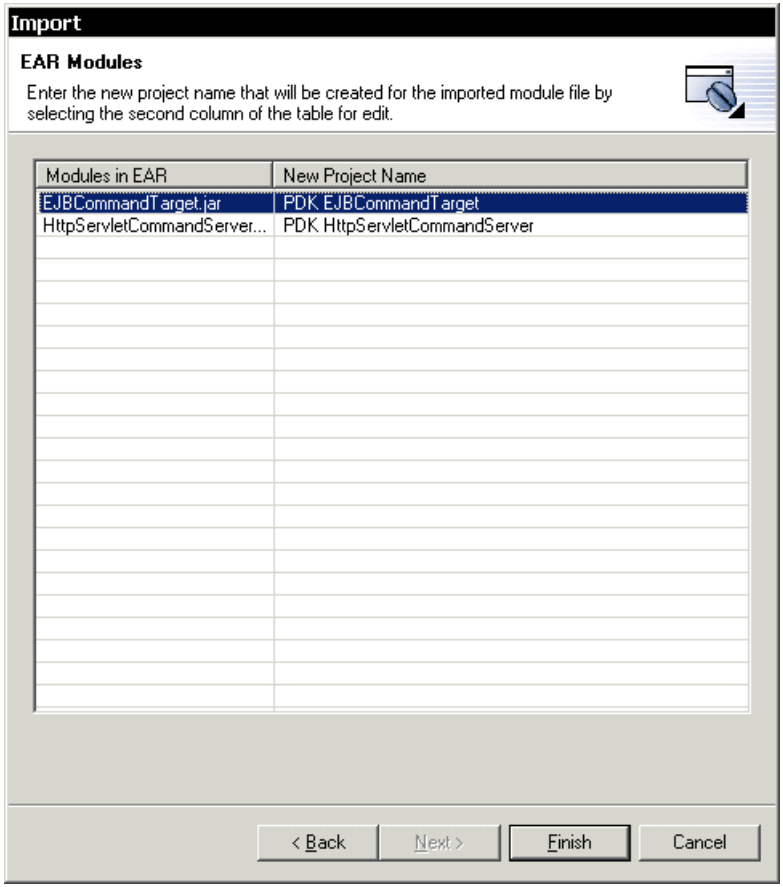


Figure 13-4 EAR modules

Click **Finish** to perform the import.



Figure 13-5 Imported projects

Now we have successfully imported Command_Servers.ear into three projects (Figure 13-5):

- PDK Command Serves (EAR project)
- PDK EJBCommandTarget (EJB project)
- PDK HttpServletCommandServer (Web project)

13.3 Fixing import problems

The import resulted in a number of errors and problems (Figure 13-6).

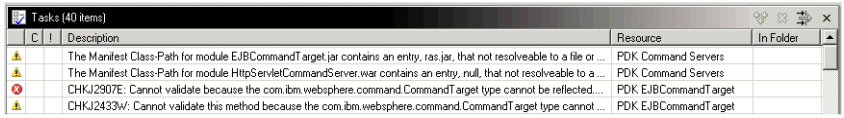

Tasks (40 items)
C
!
Description
Resource
In Folder
The Manifest Class-Path for module EJBCommandTarget.jar contains an entry, ras.jar, that not resolveable to a file or ...
PDK Command Servers
The Manifest Class-Path for module HttpServletCommandServer.war contains an entry, null, that not resolveable to a ...
PDK Command Servers
CHKJ2907E: Cannot validate because the com.ibm.websphere.command.CommandTarget type cannot be reflected....
PDK EJBCommandTarget
CHKJ2433W: Cannot validate this method because the com.ibm.websphere.command.CommandTarget type cannot ...
PDK EJBCommandTarget

Figure 13-6 Import error and warnings

One error message indicates that *com.ibm.websphere.command.CommandTarget* is missing. This would seem to be a class path problem. After checking the libraries we found that we need to add *ace.jar* to the class path. This file is provided by WebSphere. Open the property dialog of **PDK EJBCommandTarget** and add ace.jar to the Java build path. This will also solve a few other problems that were caused by the same missing jar file.

Note: ace.jar is located in the Application Developer plugin lib directory for the WebSphere runtime plugin. `C:\Program Files\IBM\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib`

To fix the error message `The Manifest Class-Path for module EJBCommandTarget.jar contains an entry, ras.jar, that is not resolvable,` open the Manifest.MF in the PDK EJB CommandTarget project and edit Manifest.MF to match the class-path in this environment by taking away *ras.jar, ace.jar,* and*EJBCommandTarget_pre.jar.* The file should now look like:

```
Manifest-Version: 1.0
Class-Path: EJBCommandTarget.imported_classes.jar
```

You need to modify the two manifest files under the bin and ejdModules.

The message `The Manifest Class-Path for module HttpServletCommandServer.war contains an entry, null, that not resolvable`, can be fixed by editing MANIFEST.MF file in the War project. Open MANIFEST.MF and remove Main-Class: null and Class-Path: null line and save (Figure 13-7). Then validate **PDK Command Servers** EAR using the **Run Validation** menu selection.

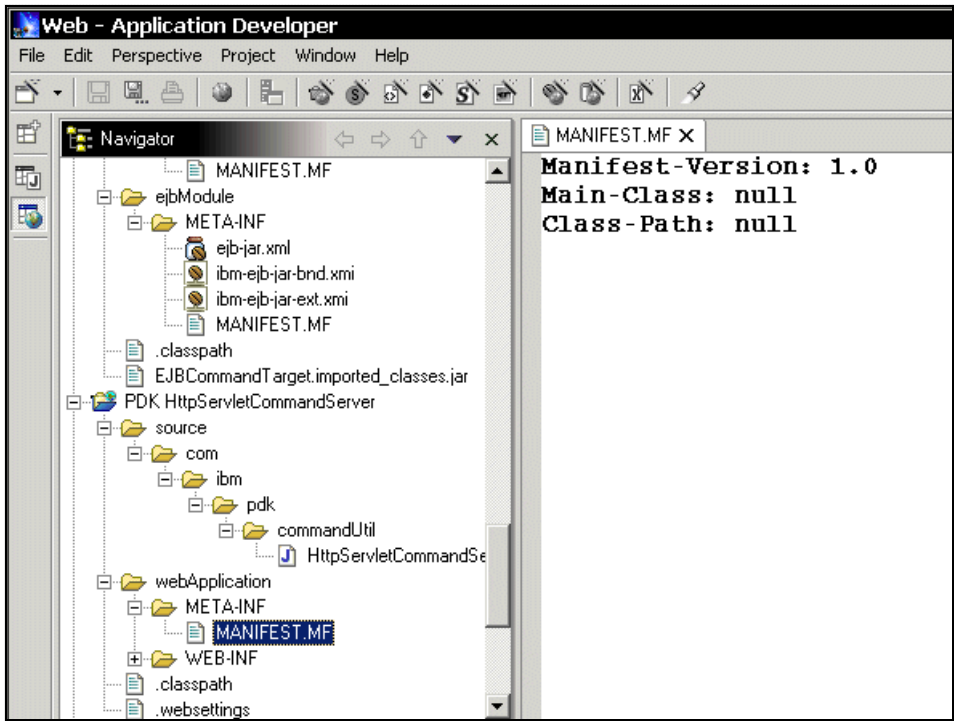


Figure 13-7 Null class problem

To resolve the other errors related to PDK HttpServletCommandServlet we need to import some missing classes. Since PDK contains a shared library we need to import that library. It is located under the artifact_x folder and its name is allshared.jar. We created a project named *PDK_Shared_Lib* and imported allshared.jar into it. Then we added a project reference to the shared library. We also modified the class path to use ace.jar and ras.jar. This completes the migration of the EAR.

13.4 Importing PDKLite

Now it is time to migrate PDKLite. PDKLite_EJB.ear is located under artifact15 folder and it contains three EJB jars, one Web application, one shared library, and descriptors (Figure 13-8).

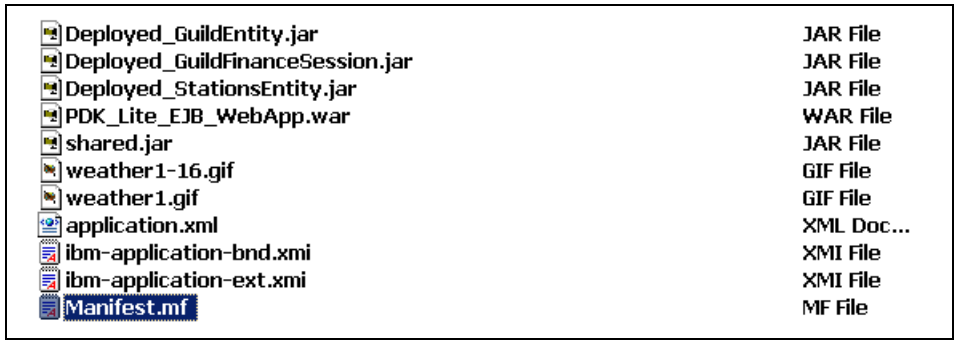


Figure 13-8 Inside of Pdk_lite_EJB.ear

We added `PDK` to each project name to distinguish it. Four projects are added to the ear project (Figure 13-9).

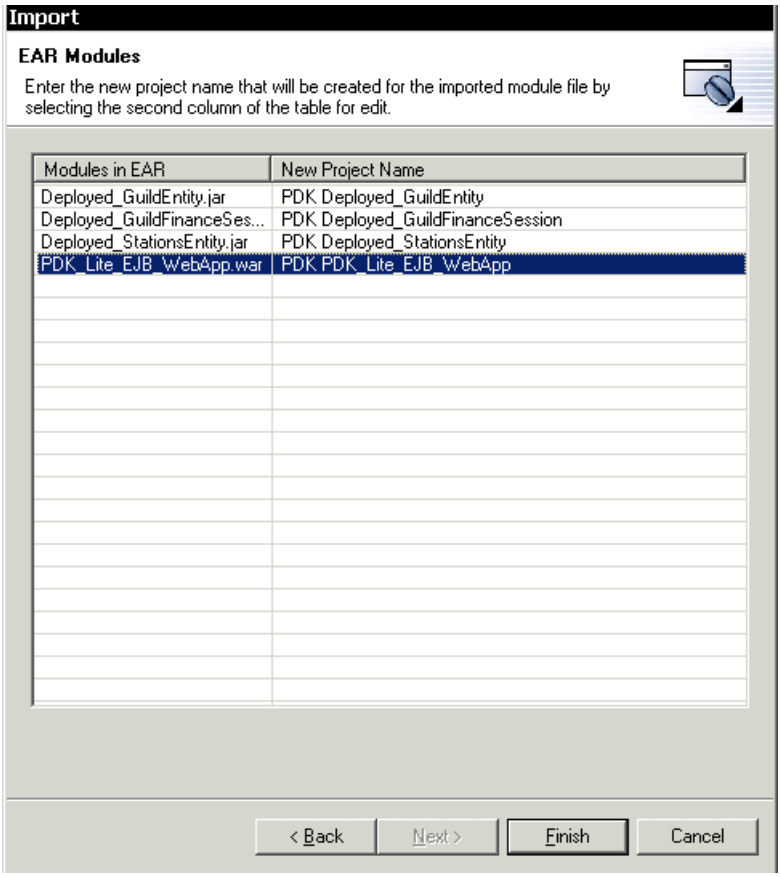
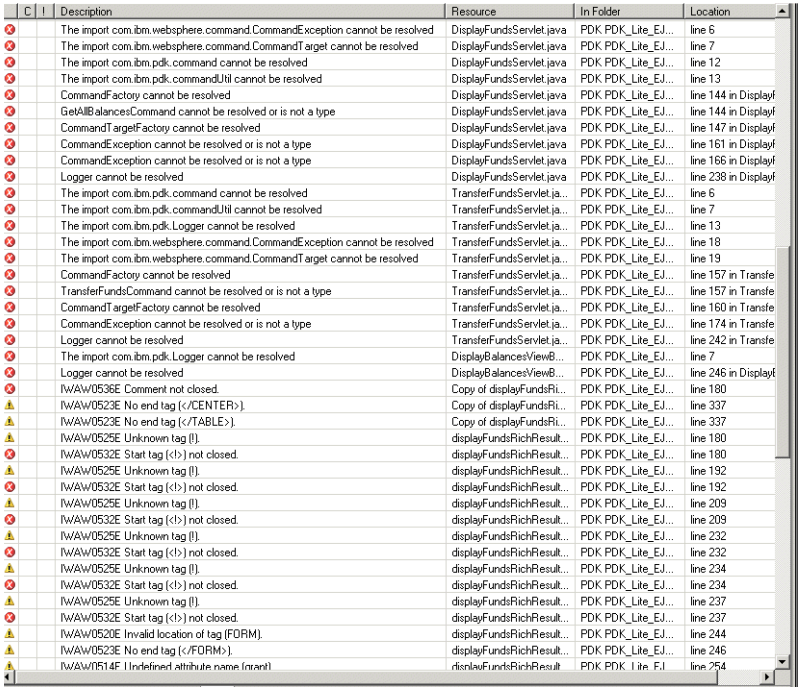


Figure 13-9 Import into projects

Click **Finish** to start the import. Once it has completed you will see a large number of errors. However, most of these are related to class paths and are straightforward to fix (Figure 13-10).



C	I	Description	Resource	In Folder	Location
✖		The import com.ibm.websphere.command.CommandException cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 6
✖		The import com.ibm.websphere.command.CommandTarget cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 7
✖		The import com.ibm.pdk.command cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 12
✖		The import com.ibm.pdk.commandUtil cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 13
✖		CommandFactory cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 144 in Displayf
✖		GetAllBalancesCommand cannot be resolved or is not a type	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 144 in Displayf
✖		CommandTargetFactory cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 147 in Displayf
✖		CommandException cannot be resolved or is not a type	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 161 in Displayf
✖		CommandException cannot be resolved or is not a type	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 166 in Displayf
✖		Logger cannot be resolved	DisplayFundsServlet.java	PDK PDK_Lite_EJ...	line 238 in Displayf
✖		The import com.ibm.pdk.command cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 6
✖		The import com.ibm.pdk.commandUtil cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 7
✖		The import com.ibm.pdk.Logger cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 13
✖		The import com.ibm.websphere.command.CommandException cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 18
✖		The import com.ibm.websphere.command.CommandTarget cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 19
✖		CommandFactory cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 157 in Transfe
✖		TransferFundsCommand cannot be resolved or is not a type	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 157 in Transfe
✖		CommandTargetFactory cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 160 in Transfe
✖		CommandException cannot be resolved or is not a type	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 174 in Transfe
✖		Logger cannot be resolved	TransferFundsServlet.ja...	PDK PDK_Lite_EJ...	line 242 in Transfe
✖		The import com.ibm.pdk.Logger cannot be resolved	DisplayBalancesViewB...	PDK PDK_Lite_EJ...	line 7
✖		Logger cannot be resolved	DisplayBalancesViewB...	PDK PDK_Lite_EJ...	line 246 in Displayf
✖		IWAW0536E Comment not closed.	Copy of displayFundsRi...	PDK PDK_Lite_EJ...	line 180
⚠		IWAW0523E No end tag (</CENTER>).	Copy of displayFundsRi...	PDK PDK_Lite_EJ...	line 337
⚠		IWAW0523E No end tag (</TABLE>).	Copy of displayFundsRi...	PDK PDK_Lite_EJ...	line 337
⚠		IWAW0525E Unknown tag (I).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 180
✖		IWAW0532E Start tag (<I>) not closed.	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 180
⚠		IWAW0525E Unknown tag (I).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 192
✖		IWAW0532E Start tag (<I>) not closed.	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 192
⚠		IWAW0525E Unknown tag (I).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 209
✖		IWAW0532E Start tag (<I>) not closed.	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 209
⚠		IWAW0525E Unknown tag (I).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 232
✖		IWAW0532E Start tag (<I>) not closed.	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 232
⚠		IWAW0525E Unknown tag (I).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 234
✖		IWAW0532E Start tag (<I>) not closed.	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 234
⚠		IWAW0525E Unknown tag (I).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 237
✖		IWAW0532E Start tag (<I>) not closed.	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 237
⚠		IWAW0520E Invalid location of tag (FORM).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 244
⚠		IWAW0523E No end tag (</FORM>).	displayFundsRichResult...	PDK PDK_Lite_EJ...	line 246
⚠		IWAW0514E Undefined attribute name (grant)	displayFundsRichResult	PDK PDK_Lite_EJ	line 254

Figure 13-10 Import errors

First of all, we modified the class path of `PDK PDK_Lite_Ejb_WebApp`. After adding `ace.jar`, `ras.jar`, and the shared library, a number of the errors were resolved.

Manifest class path error of PDK Lite ear is as same as CommandServer. Modify the mainifest.MF of the two entityBean projects and take away unresolved jar file name, then run validation for PDK Lite. This fixed another two errors.

13.5 EJB specification

We noticed that the EJBs in this package do not match the EJB1.1 Specification. The validation found the following types of errors:

- EJB has a static field that should be *final* but is not.
- EJB is using a deprecated API (*java.rmi.RemoteException*).
- An ejbCreate method of EntityBean must return the primaryKey.

To fix these problems we need to import the source code for the EJBs separately, since it is not included in the ear file. The source code is located under the `.\resources\jar\` folder in a jar file name is `pdk_ejb_project.jar`. It contains all of the source so we need ensure that we import it into the correct project. Figure 13-11 is a snapshot of GuildEntity Project.

Once you have imported the source code, you will find that the list of errors has actually grown. However, adding the class path will again get rid of most of them.

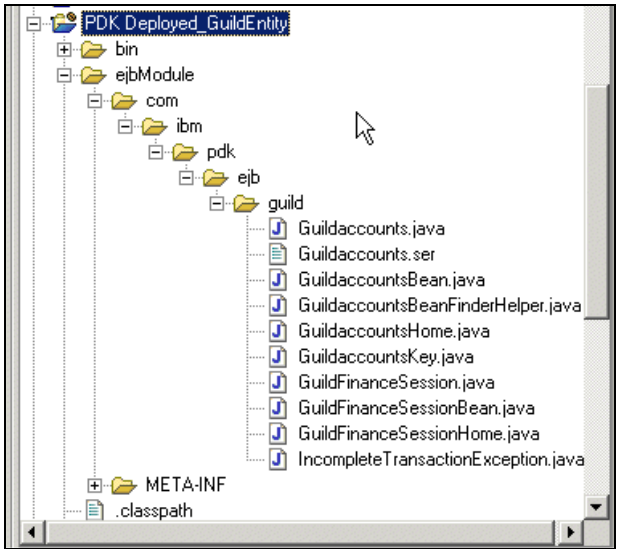


Figure 13-11 EJB source

Now you can jump to the source code by double-clicking on the problem. You need to modify as follows.

```
Old
public void ejbCreate(java.lang.String argAccount) throws
javax.ejb.CreateException, java.rmi.RemoteException {

New
public GuildaccountsKey ejbCreate(java.lang.String argAccount) throws
javax.ejb.CreateException, javax.ejb.EJBException {
```

Note: We did not touch the static field problem. To fix it we would have had to modify the Java logic. Rather than doing this, we left the deprecates in.

13.6 Fixing JSP tag error

A number of the errors are related to JSP tags. The validation has detected a problem with the following tag:

```
<! -- Rich HTML results END ->
```

To fix the problem, we modified the tag to look like:

```
<!-- Rich HTML results END -->
```

13.7 Fixing web.xml

The rest of the errors are related to web.xml. The problem is that web.xml contains relative path to the JSPs. To fix "`The JSP referenced by servlet must be a full absolute path`" problem, they should be changed to use an absolute path.

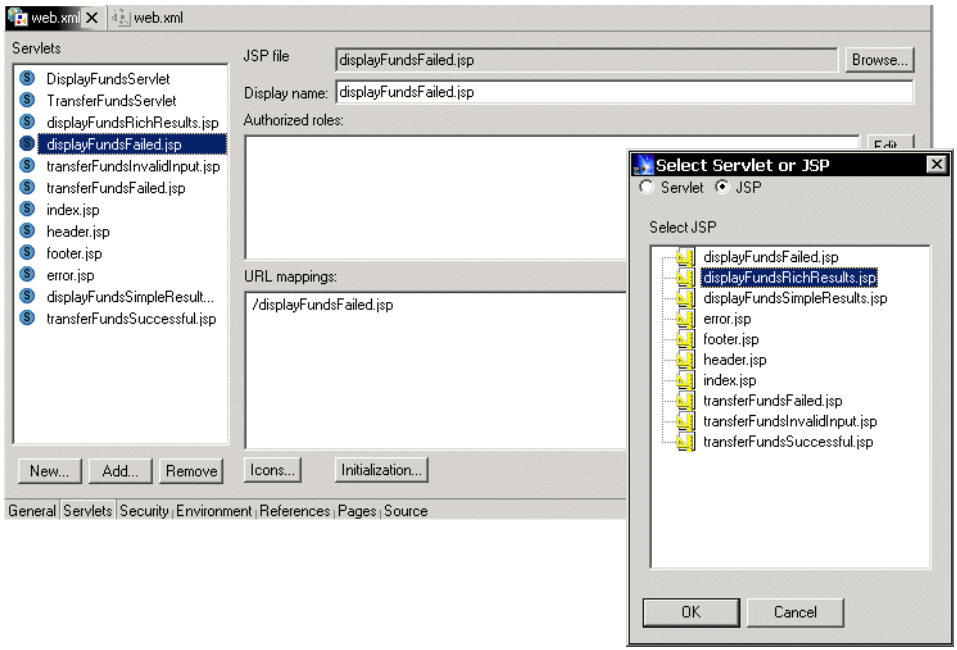


Figure 13-12 Changing relative to absolute path

Open the web.xml file and select jsp then click **Browse**. Select the JSP file and click **OK**. Once you have changed all the JPS, save the web.xml file and there should be no more errors now.

Note: You can modify the xml file by adding a '/' in front of each jsp filename.

13.8 Running PDKLite

After fixing all the errors, we tested the application to ensure that it works correctly (Figure 13-13).

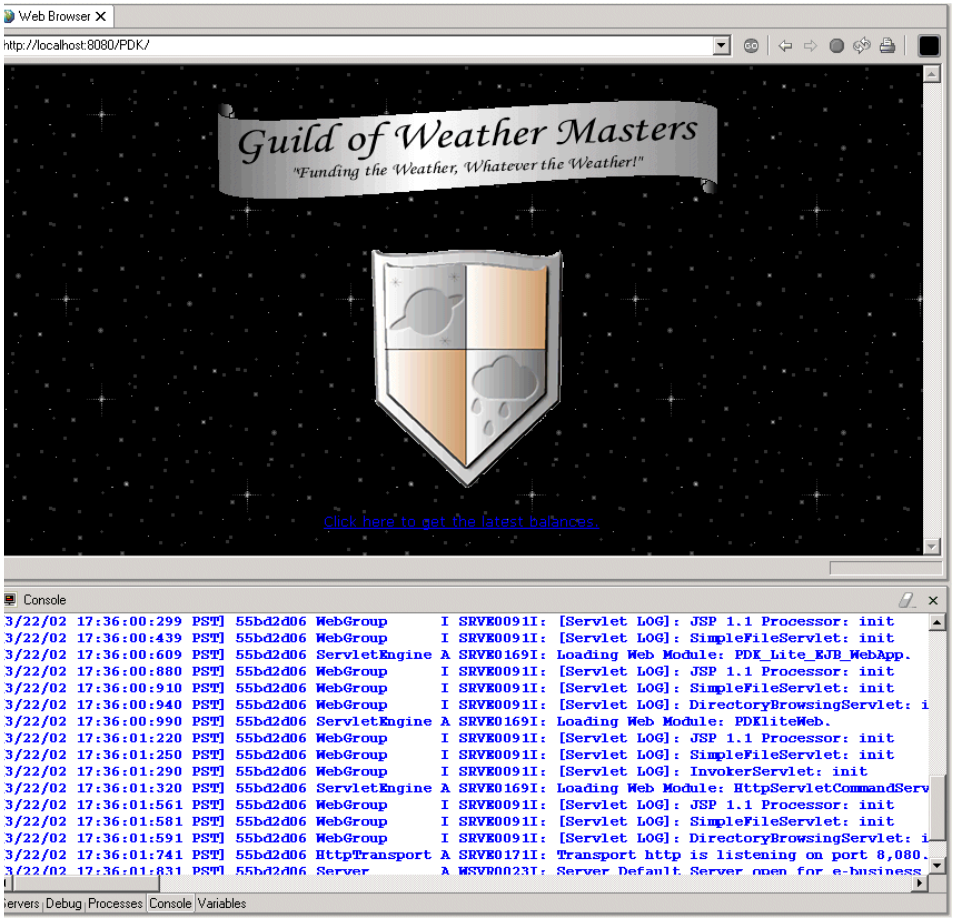

Web Browser
http://localhost:8080/PDK/
Guild of Weather Masters
"Funding the Weather, Whatever the Weather!"
Click here to get the latest balances.
Console
3/22/02 17:36:00:299 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
3/22/02 17:36:00:439 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: SimpleFileServlet: init
3/22/02 17:36:00:609 PST] 55bd2d06 ServletEngine A SRVE0169I: Loading Web Module: PDK_Lite_EJB_WebApp.
3/22/02 17:36:00:880 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
3/22/02 17:36:00:910 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: SimpleFileServlet: init
3/22/02 17:36:00:940 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: DirectoryBrowsingServlet: i
3/22/02 17:36:00:990 PST] 55bd2d06 ServletEngine A SRVE0169I: Loading Web Module: PDKliteWeb.
3/22/02 17:36:01:220 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
3/22/02 17:36:01:250 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: SimpleFileServlet: init
3/22/02 17:36:01:290 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: InvokerServlet: init
3/22/02 17:36:01:320 PST] 55bd2d06 ServletEngine A SRVE0169I: Loading Web Module: HttpServletCommandServ
3/22/02 17:36:01:561 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
3/22/02 17:36:01:581 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: SimpleFileServlet: init
3/22/02 17:36:01:591 PST] 55bd2d06 WebGroup I SRVE0091I: [Servlet LOG]: DirectoryBrowsingServlet: i
3/22/02 17:36:01:741 PST] 55bd2d06 HttpTransport A SRVE0171I: Transport http is listening on port 8,080.
3/22/02 17:36:01:831 PST] 55bd2d06 Server A WSVR0023I: Server Default_Server open for e-business
Servers Debug Processes Console Variables

Figure 13-13 PDKLite in WebSphere Test Environment

Test that all the servlets, EJBs, and JSPs are working by clicking the link in the Web browser. If everything is working, you will see the result screen (Figure 13-14).

The PDK Lite application has now been successfully migrated.

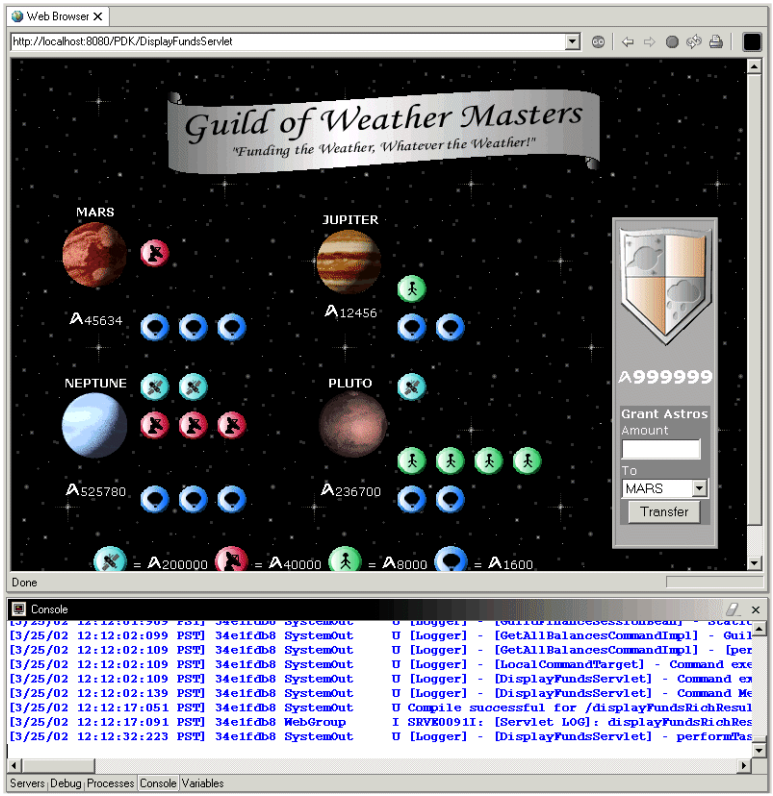

Web Browser
http://localhost:8080/PDK/DisplayFundsServlet
Guild of Weather Masters
"Funding the Weather, Whatever the Weather!"
MARS
JUPITER
A45634
A12456
NEPTUNE
PLUTO
A525780
A236700
A999999
Grant Astros
Amount
To
MARS
Transfer
= A200000
= A40000
= A8000
= A1600
Done
Console
[3/25/02 12:12:02:099 PST] 34e1fdb8 SystemOut U [Logger] - [GetAllBalancesCommandImpl] - Guil
[3/25/02 12:12:02:109 PST] 34e1fdb8 SystemOut U [Logger] - [GetAllBalancesCommandImpl] - [per
[3/25/02 12:12:02:109 PST] 34e1fdb8 SystemOut U [Logger] - [LocalCommandTarget] - Command exe
[3/25/02 12:12:02:109 PST] 34e1fdb8 SystemOut U [Logger] - [DisplayFundsServlet] - Command ex
[3/25/02 12:12:02:139 PST] 34e1fdb8 SystemOut U [Logger] - [DisplayFundsServlet] - Command Me
[3/25/02 12:12:17:051 PST] 34e1fdb8 SystemOut U Compile successful for /displayFundsRichResul
[3/25/02 12:12:17:091 PST] 34e1fdb8 WebGroup I SRVE0091I: [Servlet LOG]: displayFundsRichRes
[3/25/02 12:12:32:223 PST] 34e1fdb8 SystemOut U [Logger] - [DisplayFundsServlet] - performTas
Servers | Debug | Processes | Console | Variables

Figure 13-14 PDK Lite result screen

14

Migrating your VisualAge for Java project

Since the current version of Application Developer focuses on J2EE server-side development, deployment, and profiling, VisualAge for Java still has a wider scope that includes client development with Swing via the Visual Composition Editor, and tools such as Enterprise Access Builder for accessing back-end transaction servers that facilitate connection to many kinds of legacy systems. Even if you are currently developing these types of application, we recommend that you migrate them to Application Developer for deployment to your Web Application Server.

© Copyright IBM Corp. 2002

14.1 VisualAge for Java

In this chapter, we discuss migration from VisualAge for Java Version 3.5.3 or 4.0. If you want to migrate from an earlier version of VisualAge for Java to Application Developer, you should first migrate from your earlier version of VisualAge for Java to Version 4.0 (with enterprise beans) or Version 4.5.3 or 4.0 (without enterprise beans), before migrating to Application Developer.

14.1.1 What's new in Application Developer

The following is a summary of changes from VisualAge for Java:

- The Enterprise Java Beans specification level has changed from 1.0 to 1.1.
- The level of the Java 2 platform that is supported has changed from 1.2 to 1.3.
- The VisualAge for Java XML tools have been replaced by WebSphere Studio Application Developer XML tools.
- VisualAge for Java version control has been replaced by WebSphere Studio Workbench support for Source Code Management plug-ins.
- The VisualAge for Java project concept has been replaced by multiple types in WebSphere Studio Application Developer projects.

14.1.2 Coexisting with VisualAge for Java

Some VisualAge for Java development tools do not have a corresponding tool in Application Developer. If you want, you can continue to develop some pieces of your application using VisualAge for Java and then deploy your code to Application Developer in exactly the same way that you would deploy from VisualAge for Java to WebSphere Application Server. This allows you to keep all of your application resources together in the Application Developer Workbench for eventual deployment to a test or production server.

14.1.3 Project structure

There is no support for the bulk migration of versioned projects and resources from the VisualAge for Java repository. You can only migrate projects and resources that are in your VisualAge for Java workspace. If you want to migrate a versioned copy of a project or resource into Application Developer, you must add it to your VisualAge for Java workspace and then migrate it.

If your project contains more than one kind of nature, such as Java, EJB, or Web resources, the project should split up your data into different Application Developer projects based on their nature (Figure 14-1). This step should be done by yourself.

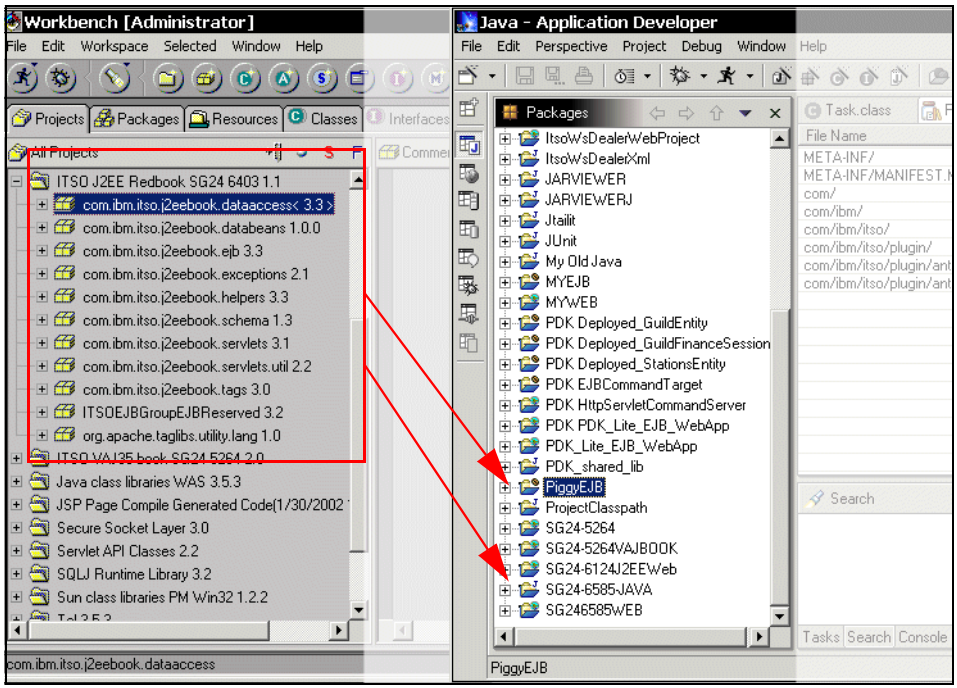


Figure 14-1 Several natures in VisualAge for Java

14.1.4 Visual Composition Editor (VCE)

Application Developer does not have the capability to edit your VCE application, but can import your VCE application into the workbench. To run the VCE application, export the project as a jar file and import into the workbench. If you are planning to modify your VCE application later, you should not edit the application on the workbench. VCE application is keeping VCE specific data as a comment to reconstruct VCE data when VCE is reopened. Even if you edit the code, VCE throws them away and reloads from the data.

Note: The user code section is not affected by this modification. If you only change the code in the user code section, you can go back to VCE to edit your application graphically.

To migrate your VCE application, export your project as a jar file from VisualAge for Java workbench. Then create a new project in the Application Developer Workbench and import the jar file.

14.1.5 Source location

In Application Developer, each project type has different source location. Since VisualAge for Java projects have a single location you have to handle with care.

Type	**Location** (all locations are under the project directory)
Java	.(root)
Web	source
Web(classes)	webApplication/WEB-INF/classes
Web(resouces)	webApplication
EJB	ejbModule
EJB(binary)	bin
Plug-in	<plug-in name> plugin

14.2 Migrating a Web project

For our example we are assuming that you have a project that was created using the **SG24-5264-01 Programming with VisualAge for Java 3.5** redbook. The project name is **ITSO VAJ35 Book SG24 5264** (Figure 14-2). Use the export function in VisualAge for Java to export the whole project (do not forget to export the source files, unless you will not be modifying the source code). You can't export it as a repository file. You should choose the Jar or file structure option. We exported the SG245264 Project as SG245264.jar.

Note: We used the samples from **SG24-5264 Programming with VisualAge for Java 3.5 Redbook**, and the jar file is included in this book.

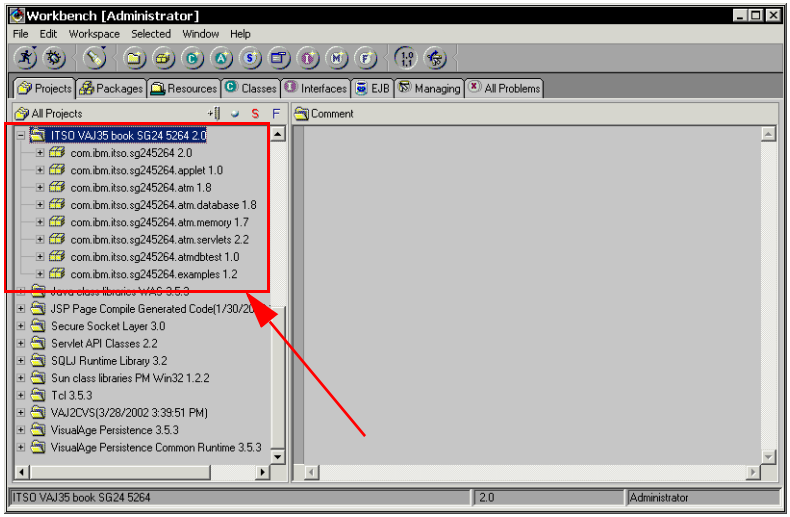


Figure 14-2 Exporting SG24-5264 project

In the export SmartGuide, select Jar file and click **Next**. Then set the file name as `sg245264.jar` and select all `.java` files and resources (Figure 14-3).

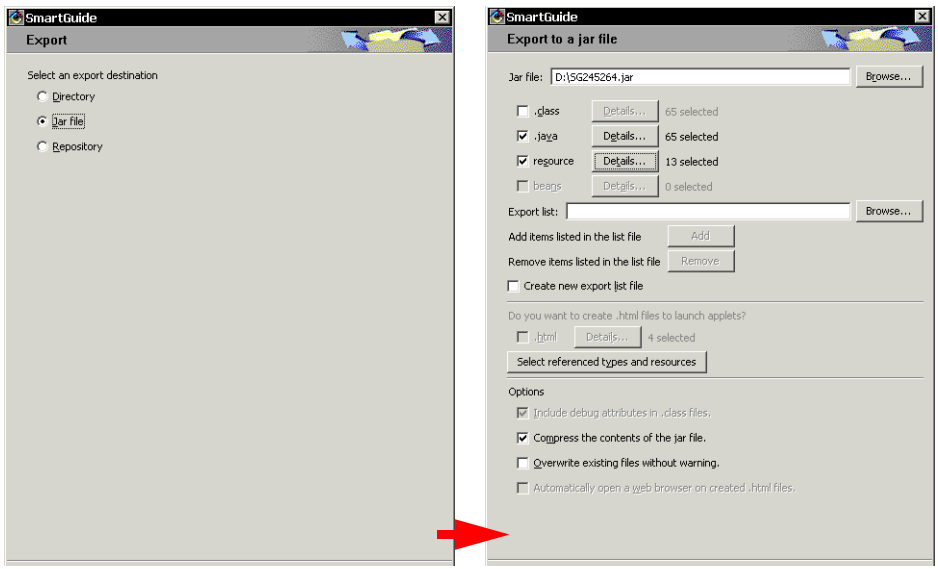


Figure 14-3 Exporting as a jar file

14.3 Setting up your project

Since SG25-5264 sample is a Web application, we need to have a Web project defined in Application Developer. Open the Application Developer Workbench and create a Web project. We created the **SG24-5264** project to hold the SG24-5264 Java sample codes. We named the EAR SG24-5264EAR. Once the Web project has been created, the workbench will switch to the Web perspective (Figure 14-4).

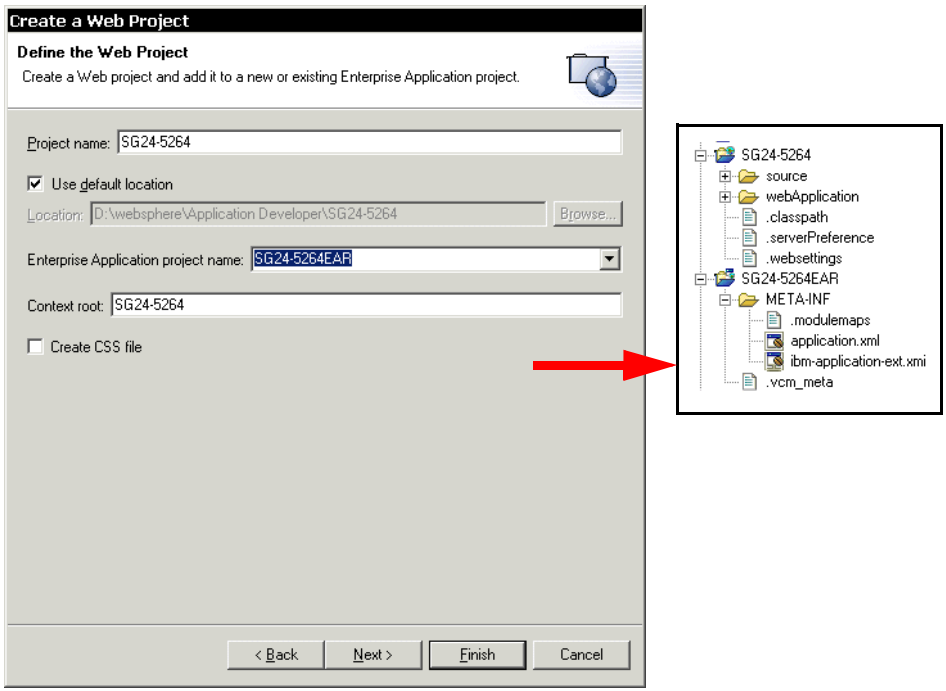


Figure 14-4 Creating a Web project

14.4 Importing a jar file

As we mentioned, all Java code should be saved in the source folder. Click the source folder to select the target for the source code import. Then use the import wizard and select **ZIP** to import **SG245264.jar** (Figure 14-5). We want to import the Java code but not the JSPs. To import only the Java source select root, click the **Select types** button, and select ***.java**. Then click **Finish** to start the import. All the Java code is imported and compiled into classes. We will discuss later how to migrate Web applications. Once all the code has been imported, switch to the Java perspective to see it.

Note: Use the Java perspective to edit or fix errors in your Java code. All errors will be shown in the Task view.

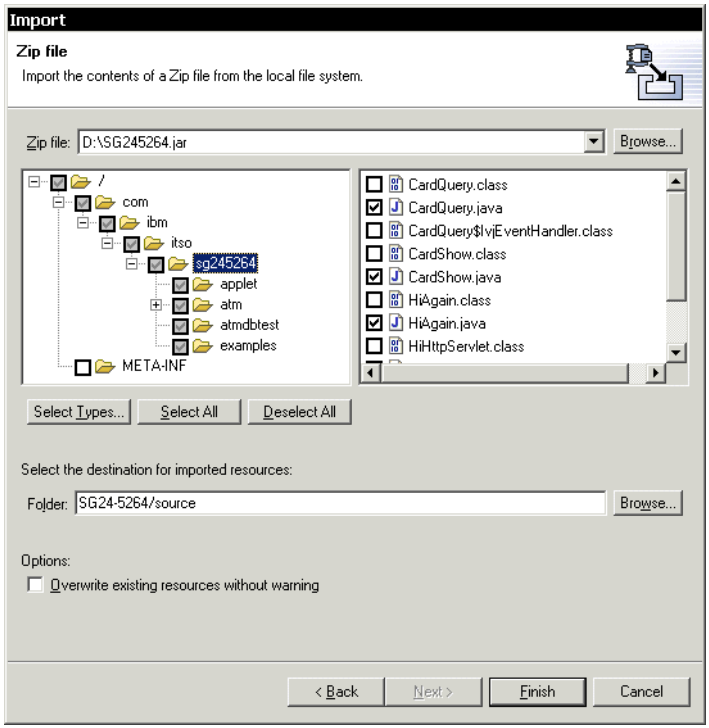


Figure 14-5 Importing Java code

14.4.1 Fixing the class path

There are now a number of errors in the project. Double-click `CardQuery.java` to open the code (Figure 14-6). Since this is a VCE application, it uses the `com.ibm.ivj.uibean` package. To fix the problem you need to add the VisualAge for Java specific class library. In VisualAge for Java, you set your project class path in the Resources pages of the Options window. After you have migrated your projects into Application Developer, you can setup up your project's class path in the project properties window. You can also need to set the class path variables in the Preferences window. See "Defining Java class path variables" on page 24.

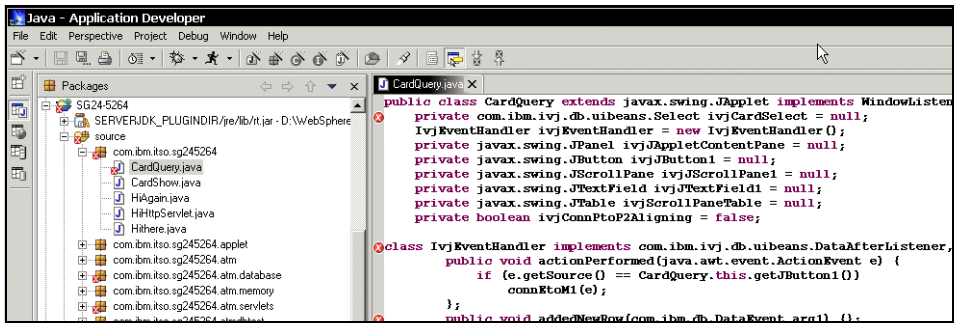


Figure 14-6 Result from import

Other missing classes belong to the IBM Data access bean library. To resolve these, you need to add `ivjdab.jar` (which is included with Application Developer). Open the property dialog and add a variable as follows (Figure 14-7).

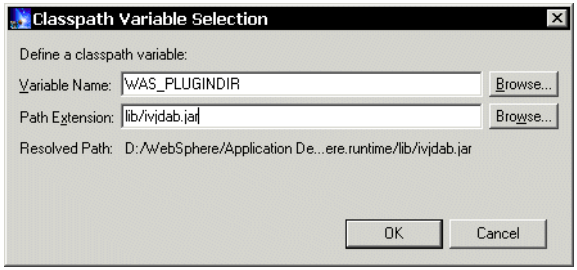


Figure 14-7 Adding a class path variable

Another required library is SQLJ which is demonstrated in the `com.ibm.itso.sg245264.example` package. To use SQLJ, you need to add runtime.zip to the class path. The `runtime.zip` is located in the `./Program Files]\SQLLIB\java` directory, if you are using DB2. Also, you need to add `db2java.zip` to get access to the JDBC driver class. (This problem is not shown in the Task view since it is only detected at runtime.) Figure 14-8 shows all the required libraries.



Figure 14-8 Required libraries

14.4.2 Inner classes

Migrating inner classes can also cause problems. For example, `com.ibm.itso.sg245264.atm.database.Card.java` contains the CardPK class inside it. Check the Task view. You will find the following error: **"**`The type Card$CardPK is an incorrectly specified nested type; replace the '$' with '.'`. To correct this, just double-click on the error, and change the '$' to '.'.

```
//wrong code
tempCard = new Card((new Card$CardPK(tmpcardnum)),tmpcardowner, tmpPIN );
//corrected code
tempCard = new Card((new Card.CardPK(tmpcardnum)),tmpcardowner, tmpPIN );
```

The same problem applies to `ShowATMServlet.java: 'ATM$ATMState cannot be resolved...'`. ATMState is an inner class of ATM class, so change '$' to '.' to fix this problem.

14.4.3 Deprecated methods

The Task view shows all deprecated methods as warnings. To change your code, you need to consult the Java class library documentation. Each deprecated method has an alternative solution. It is a good idea to fix these at this time.

In our case, CardQuery is using one deprecated method. To fix it, change it as follows.

```
//Deprecated
//getJScrollPane1().getViewport().setBackingStoreEnabled(true);

//New
getJScrollPane1().getViewport().setScrollMode(
                              javax.swing.JViewport.BACKINGSTORE_SCROLL_MODE);
```

Most of servlets are using getValue method of HttpSession class, which is deprecated. The alternative method is getAttribute.

```
//deprecated
ATM userATM = (ATM)session.getValue("userATM");
//new
ATM userATM = (ATM)session.getAttribute("userATM");
```

Also putValue should be replaced by setAttribute method.

```
//deprecated
session.putValue("userATM", userATM);

//new
session.setAttribute("userATM", userATM);
```

14.5 Running the samples

The SG24-5264 project should now be ready to run. You can run the Java application. To run it, click the class, for example QueryCard.java, then click the **Run** icon  and select **Java Application**.

Note: CardQuery is an applet but also has a main method to run as an application generated by the VisualAge SmartGuide.

To run a servlet, you must go into the webApplication folder and find the class instead of source. Then select **Run on Server** from the context menu to launch. A server instance will be generated automatically and the application will run in the Application Developer Web browser. Figure 14-9 shows that HiHttpServlet was invoked.

14.5.1 WTE configuration

In VisualAge for Java, the WebSphere Test Environment and WebSphere Application Server runtime settings are in various files in the WTE directory. In Application Developer, you can configure the properties using WebSphere V4.0 Test Environment which is located under the Server perspective, in the Server configuration window. Web application settings in file defailt_app.webapp that you have customized, should be migrated to the web.xml file under your Web project, in the WEB-INF folder.

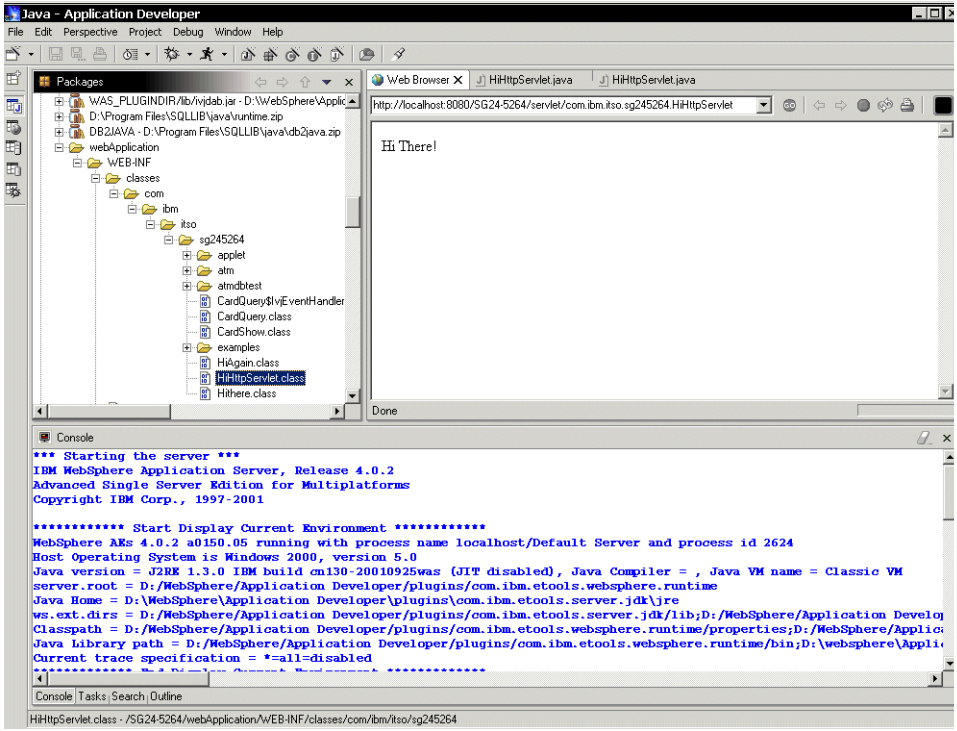


Figure 14-9 Running a servlet

14.6 Importing JSPs and HTML files

Since JSPs and HTMLs are only resources in VisualAge for Java environment, the migration is very similar to the PDK migration discussed in Chapter 13, “Migrating your Java classes” on page 325. Since these are not packaged in a WAR file, you will need to edit the web.xml file to link the servlets and JSPs together.

All the required files are located in `SG245264.jar`. Select webApplication folder and import them by selecting HTML and JSP only (use select types).

Then open the `web.xml` file. Now all the servlets and JSPs have to be defined (Figure 14-10).

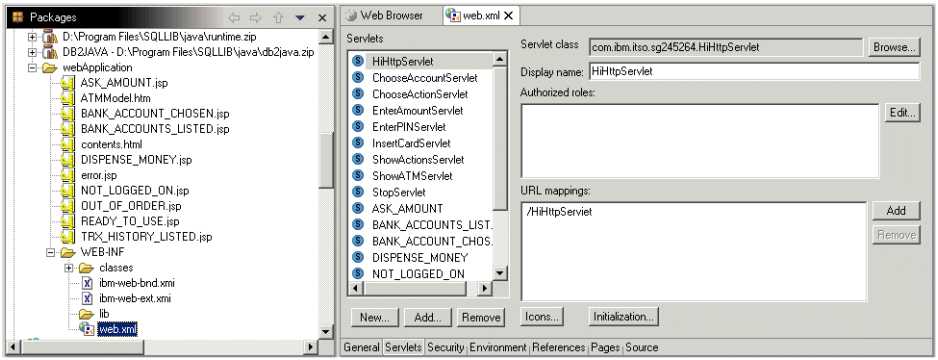


Figure 14-10 Editing servlets/JSPs definition

14.6.1 Changing the invoker to alias style

There are several broken link errors in the Task view. You will notice that the servlets in the `SG245264` sample are using the invoker servlet. Each link has the following style:

```
http://localhost/servlet/com.ibm.itso.sg245264.atm.servlets.XxxxServlet
```

The tag /servlet/ is the alias of the invoker servlet. The invoker servlet forwards the request to the target servlet. This style is old and not safe so we will change the links to use the alias method instead. We have already added all of the servlets and JSPs to `web.xml`, so all of them will now have an alias. All we need to do is to modify the JSPs to invoke the servlet by alias.

For example, `ASK_AMONT.jsp` has the following statement:

```
<FORM METHOD='POST'
ACTION='/servlet/com.ibm.itso.sg245264.atm.servlets.EnterAmountServlet'>
```

We modified this to:

```
<FORM METHOD='POST' ACTION='/SG24-5264/atm/EnterAmountServlet'>
```

Note: SG24-5264 is the context root of this project, and we mapped *EnterAmountServlet* as */atm/EnterAmountServlet*.

14.6.2 HTML tags

In *Programming with VisualAge for Java 3.5*, SG24-5264, a number of HTML and JSP files were created. These were manually written, and the Application Developer validation complains about how the <FORM>..</FORM> tag is inside of <P> ... </P>. To resolve this problem we moved the FORM tag to outside of the paragraph.

Before:

```
<P>Please enter the amount to withdraw now. <BR>
<FORM METHOD='POST'
ACTION='/servlet/com.ibm.itso.sg245264.atm.servlets.EnterAmountServlet'>
Amount: <INPUT TYPE=TEXT NAME='amount' SIZE=10>
<INPUT TYPE=SUBMIT VALUE='Next'>
</FORM> </P>
```

After:

```
<P>Please enter the amount to withdraw now. </P><BR>
<FORM METHOD='POST' ACTION='/SG24-5264/atm/EnterAmountServlet'>
Amount: <INPUT TYPE=TEXT NAME='amount' SIZE=10>
<INPUT TYPE=SUBMIT VALUE='Next'>
</FORM>
```

14.6.3 JSP tags

Since this sample was created using XML style JSP tags, we only got one error when importing the JSP. It concerns the jsp:root tag. When we imported the JSP file, Application Developer modified it as follows.

Original:

```
<jsp:root
   xmlns:jsp="http://java.sun.com/products/jsp/dtd/jsp_1_0.dtd">
<jsp:directive.page
   errorPage="/itsojsp/error.jsp"
/>
.
.
.

<P>
<A HREF='/servlet/com.ibm.itso.sg245264.atm.servlets.StopServlet'>Restart</A>
</P>
</BODY></HTML>
</jsp:root>
```

Modified by Application Developer:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<jsp:root
   xmlns:jsp="http://java.sun.com/products/jsp/dtd/jsp_1_0.dtd">
<jsp:directive.page
   errorPage="/itsojsp/error.jsp"
/>
.
.
.
<P>
<A HREF='/SG24-5264/atm/StopServlet'>Restart</A>
</P>
</BODY></HTML>
</jsp:root><HTML></HTML>
```

Because of the support level of `jsp:root tag`, Application Developer is not able to recognize it. To fix this we cleaned up the JSP by taking away the jsp:root tag along with the superflous `HTML` tag.

What is the jsp:root tag?: A JSP page in XML syntax has jsp:root as its root element. The jsp:root element is responsible for specifying the appropriate namespaces available to the document. The JSP root element has two attributes, the xmlns and the version attribute.

15

Other migration tips

This chapter contains some other tips about migrating your applications to Application Developer.

- SQLJ and stored procedures
- Visual Age Persistence Builder
- Enterprise JavaBeans
- Sharing a CVS repository

© Copyright IBM Corp. 2002

15.1 SQLJ and stored procedures

The Structured Query Language Java (SQLJ) feature of VisualAge for Java provides a standard way to embed SQL statements in Java programs. As we discussed in “Fixing the class path” on page 345, you need to add two libraries the project class path to get access to the database classes: `db2java.zip` and runtime.zip. (You can find sqlj-runtime.jar in the Visualage for Java folder.) When your application has been completed it can be deployed to the target environment (these environments must have access to the SQLJ run-time ZIP files.)

The DB2 Stored Procedure Builder feature of VisualAge for Java is a graphical application that supports the rapid development of DB2 stored procedures that run on a database server. You can export the files you create with this feature from VisualAge for Java to Application Developer for use with your database application.

15.2 VisualAge Persistence Builder

The Persistence Builder feature of VisualAge for Java provides a framework for building robust, scalable persistence support for applications. You need to use a library of VisualAge Persistence Builder ivjpb353.jar (or ivjpb353wsa353.jar especially for WebSphere Application Server AE) which is located in the VisualAge `.\Java\eab\runtime35` folder.

When you import the VisualAge Persistence Builder library into your project in the workbench, you must select File system, not Zip file, in the Import wizard in order to ensure the JAR file is not expanded when it is imported. Then add the JAR to the Java build path of the project. The Persistence Builder run-time JAR will eventually be deployed as a JAR along with your application. You then need to modify each Persistence Builder Business Object HomeImpl class to conform to the EJB1.1 specification, and add the following method to each class.

```
public javax.ejb.HomeHandle getHomeHandle() { return null; }.
```

If the function that you developed in Persistence Builder requires changes, make your changes in VisualAge for Java, regenerate your code in Persistence Builder, re-export the code, and re-import it into the Application Developer Workbench. In this case, you need to add the new `getHomeHandle()` method to any redeployed Persistence Builder BusinessObject HomeImpl class each time.

When your application is complete, you can deploy it to your target environments the same as any other application developed in Application Developer application. (You must include the Persistence Builder run-time JAR.)

15.3 Enterprise JavaBeans

To migrate your EJBs, you have to export EJBs using VisualAge for Java Version 4 export EJB 1.1 feature. Once you have exported your EJBs in the EJB1.1 jar format, you can import them into the workbench (Figure 15-1).

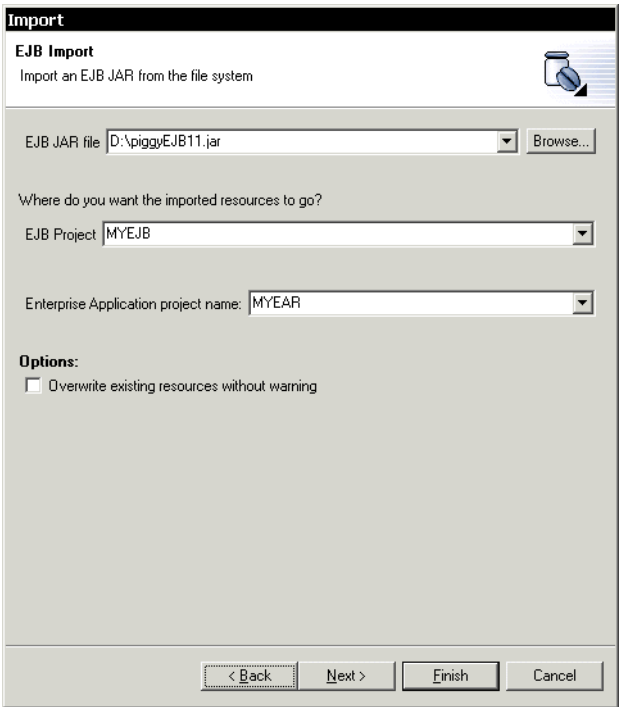


Figure 15-1 Import the EJB

Both the EJB project and the EAR project are created by the import wizard. In this case, we used the sample from the redbook *Programming J2EE APIs with WebSphere Advanced*, SG24-6124. Since the sample is targeting WebSphere Application Server 3.5.3, the EJB is not fully compatible with EJB1.1.

First of all, we export the EJB as an EJB1.1 jar file, along with the other class files that are used by the EJB. Then we import the EJB, creating the EAR and EJB projects. After that, we create a Web project and import the class files. In the Web project, we removed the superfluous EJB and schema packages, and moved all source files into the source folder (Figure 15-2). Since both projects are referencing each other, the EJB project should be accessible from the Web project, and the Web project should be accessible from the EJB project. We need to modify the Java build path so these projects are able to access each other.

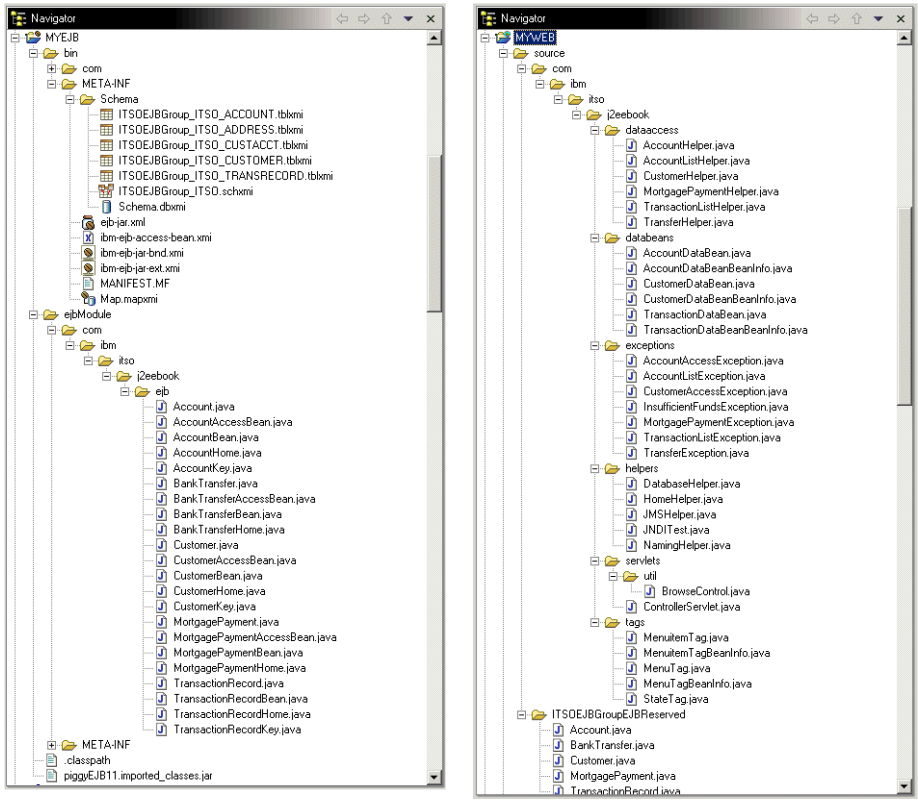


Figure 15-2 Imported projects

After modifying the class path, there is still one error and a number of warnings remaining. Most of warnings relate to `java.rmi.RemoteException` and to the `PrimaryKey` of the `ejbCreate()` method.

The problem is that the return type is not is not serialized:

```
/**
 *
 * @return com.ibm.itso.j2eebook.ejb.CustomerAddress
 * @exception String The exception description.
 */
com.ibm.itso.j2eebook.ejb.CustomerAddress getAddress()
                                    throws java.rmi.RemoteException;
```

We looked at the definition of CustomerAddress class, and confirmed it is a serializable object:

```
public class CustomerAddress implements java.io.Serializable {
```

We need to tell the validator that it is actually serializable, so we added an @serial tag in the comment as follows.

```
/**
 *
 * @return com.ibm.itso.j2eebook.ejb.CustomerAddress
 * @exception String The exception description.
 * @serial:getAddress()
 */
com.ibm.itso.j2eebook.ejb.CustomerAddress getAddress()
                                        throws java.rmi.RemoteException;
```

Once Application Developer recognizes that is a serializable, it won't complain again, even if we were to remove the `@serial` tag.

Now we need to rebuild the project to remove all warnings related to the CustomerAddress.

The rest of warnings are categorized as follows:

- Use `javax.ejb.EJBException` instead of java.rmi.RemoteException in an ejbCreate method.
- An ejbCreate should return a `primaryKey` instead of `void`.
- A matched `ejbPostCreate` with `ejbCreate` must exist.

The task list will be cleared when all the deprecates above have been fixed. Now you can generate the code from the J2EE perspective.

15.4 Using the CVS repository from VisualAge for Java

VisualAge for Java can use the CVS repository via the VAJ2CVS utility. The `VAJ2CVS` tool is freeware and is available at `http://vaj2cvs.sourceforge.net`. To use this tool, you need to modify the modules file in the `CVSREPOSITORY` folder. Once it has configured, you can login to CVS from VisualAge for Java and use CVS as its SCM.

Note: You can check in / check out the projects which were created in the VisualAge Workbench and register them in the CVS repository. You cannot add a project that was created in the Application Developer Workbench into the VisualAge Workbench.

Figure 15-3 shows the CVS menu.

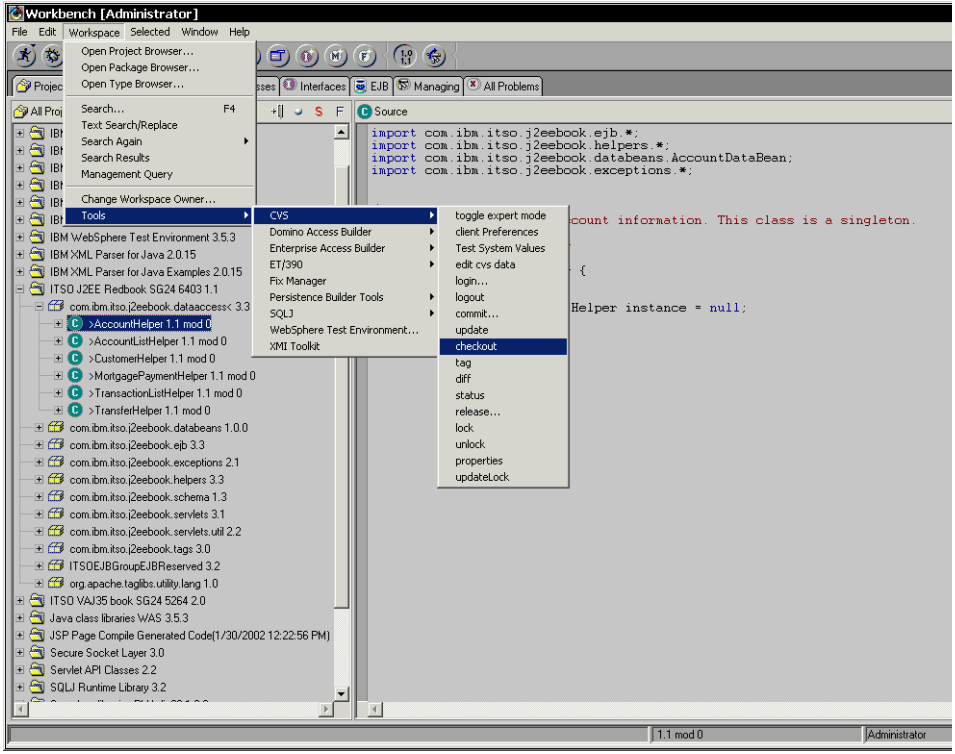


Figure 15-3 Using CVS from VisualAge for Java

VAJ2CVS uses the VisualAge for Java IDE Remote Tool and Swing package. Figure 15-4 shows the status list of each class. Once you add and commit your files to CVS, you can access them from the Application Developer Workbench (Figure 15-5).

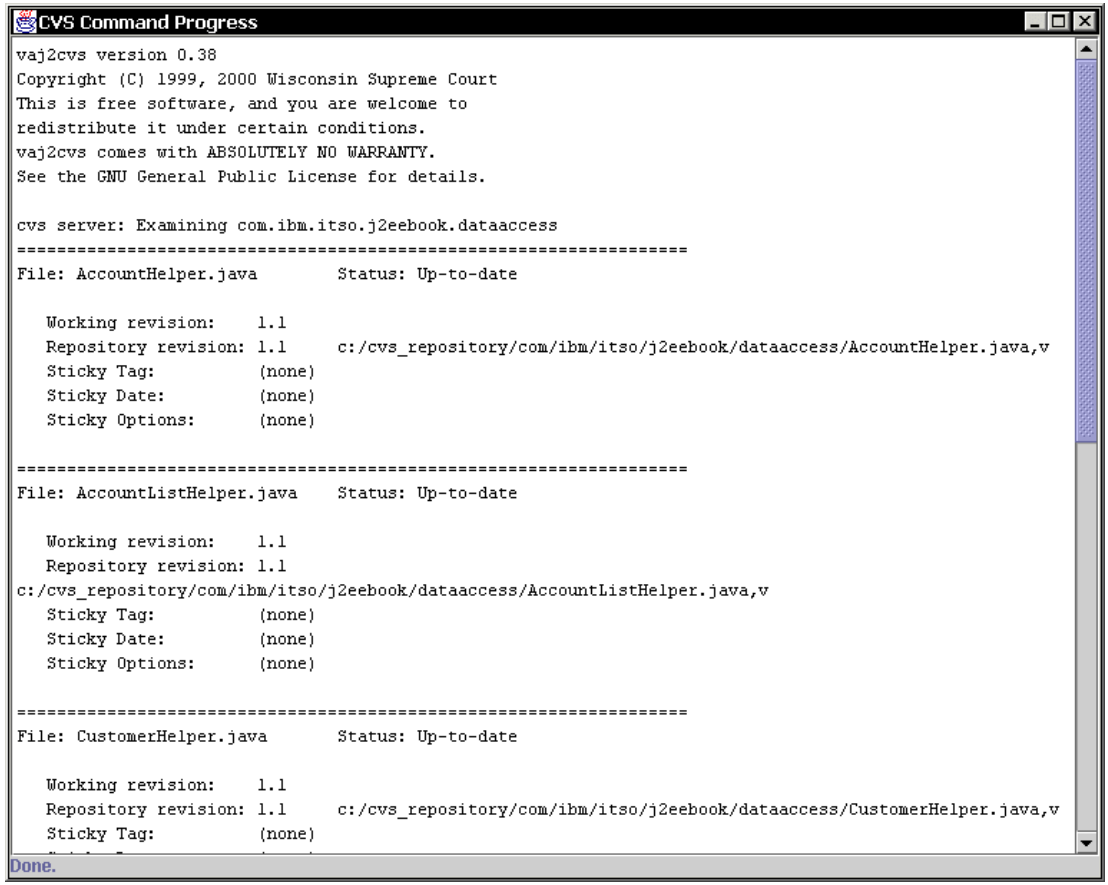


Figure 15-4 Class status in CVS

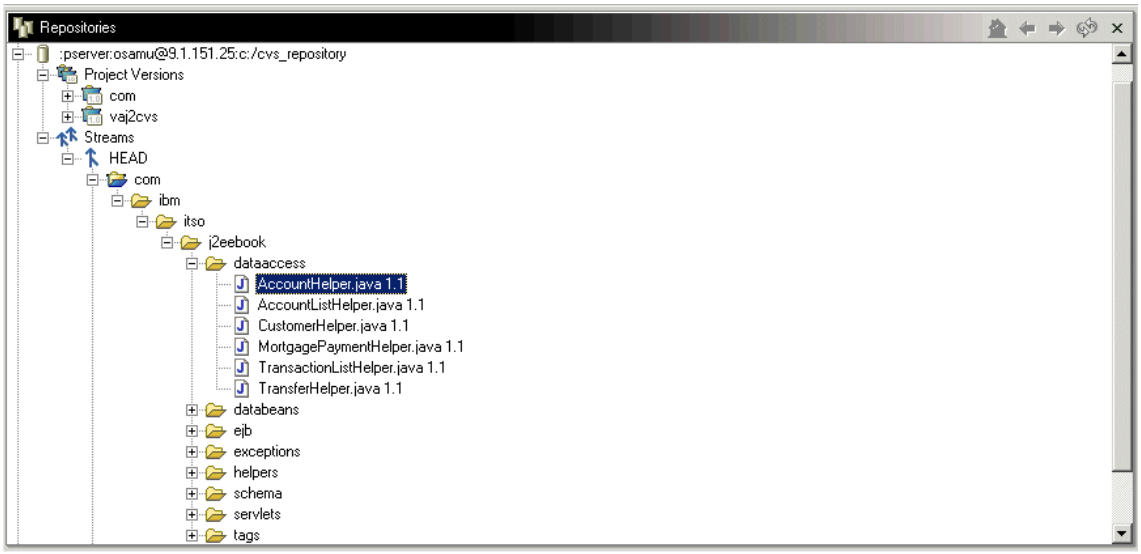


Figure 15-5 Accessing the VisualAge Repository in Application Developer

Part 5

Testing and deploying your Web applications

Part 5 describes how to test your Web application and deploy it to WebSphere Application Server.

© Copyright IBM Corp. 2002

16

Server instances and server configurations

Application Developer provides the capability for testing, debugging, and deploying your Web applications in local and remote environments.

- WebSphere Application Server AEs (developer edition, that contains the same code as AEs), is built into the Application Developer.
- WebSphere Application Server AEs (a single server edition) can be installed on the same or on a remote machine. A remote server can be started through the IBM Agent Controller, which must be installed on the machine where the server runs.
- Apache Tomcat is included in Application Developer and can only run on the local machine. EJB's are not supported in Tomcat.

To test or debug a Web application in Application Developer it must be published or deployed to the selected server. You achieve that by installing the owning EAR project file into the application server. Then the server can be started and the Web application can be tested in a Web browser.

This chapter describes: server tools, server perspective, server instances and server configurations, and local and remote servers.

© Copyright IBM Corp. 2002

16.1 Server Tools feature

Application Developer provides the Server Tools feature to support you in testing your Web projects. It provides test environments where you can test JSP files, servlets, HTML files, and EJBs. It also provides a tool called the EJB test client where you can test your EJB components.

The Server Tools feature uses *server instances* and *server configurations* to test and deploy your projects. Server instances identify servers where you can test your projects. Server configurations contain setup information.

You can either have the Server Tools create the server instances and server configurations automatically for you, or you can create them using a Server Tools wizard.

The Server Tools feature allows you to create server instances and server configurations that can run resources from the following types of projects:

- **Web projects**: Which may contain JSP files, HTML files, servlets, and beans.
- **EJB projects**: Which contain EJB beans.
- **Enterprise Application projects**: Which may contain Java Archive (JAR) files or Web Archive (WAR) files or both, and pointers to other Web or EJB projects.

You can use the Server Tools views to manage the server instances and server configurations. When running the Application Developer test environment, the server is running against the resources that are in your workspace. Therefore, you can add, change, or remove a resource from the Web project and the server is able to pick up these changes without having to be restarted or republished.

16.1.1 Supported run-time environments

The Server Tools feature allows you to test your applications in different run-time environments that can be installed locally or remotely.

- The Server Tools feature includes a local copy of the full IBM WebSphere Application Server Advanced Single Server Edition for Multi platforms (WebSphere Application Server AEs) run-time environment, where you can test Web projects, EJB projects, and Enterprise Application projects.
- You can also test on a remote copy of the WebSphere Application Server AEs product. To do this, you must install the following products on your remote machine:
  - IBM WebSphere Application Server Advanced Single Server Edition for Multi platforms

  - IBM Agent Controller
- The Server Tools feature also supports the Apache Tomcat run-time environment, running locally.

  Important: With Apache Tomcat, you can test only Web projects that contain servlets and JSPs.

- A run-time environment called the TCP/IP Monitoring Server is also packaged with the Server Tools feature. This is a simple server that monitors all the requests and responses between the Web browser and an application server. It also monitors TCP/IP activity. This run-time environment can only be run locally. You cannot publish projects to the TCP/IP Monitoring Server as well.

Table 16-1 What projects can run in which environment

	WebSphere Application Server	**Tomcat**
Enterprise Application projects	**X**	
EJB projects	**X**	
Web projects	**X**	**X**

16.1.2 WebSphere Test Environment benefits

The Server Tools feature contains the complete run-time environment of the IBM WebSphere Application Server AEs. In the Server Tools feature, this environment is called *WebSphere Test Environment*.

WebSphere Test Environment offers the following benefits:

- Standalone all-in-one testing.
- No dependency on WebSphere Application Server installation or availability.
- No dependency on an external database even when entity bean support is required.
- Provides the ability to debug running server-side code.
- Supports configuring multiple Web applications.
- Supports multiple servers that can be configured and run at the same time.
- Provides access to the profiling feature that is available in the Application Developer.
- Provides the ability to version Server Tools server configurations.

- Provides an EJB test client.
- Provides access to the WebSphere Application Server Administration Client.

16.2 Server perspective

Application Developer provides a perspective where you can perform the relevant server configuration and management tasks. This is the Server perspective.

This perspective was introduced in “Server perspective” on page 74. Figure 16-1 shows the default layout of the perspective.

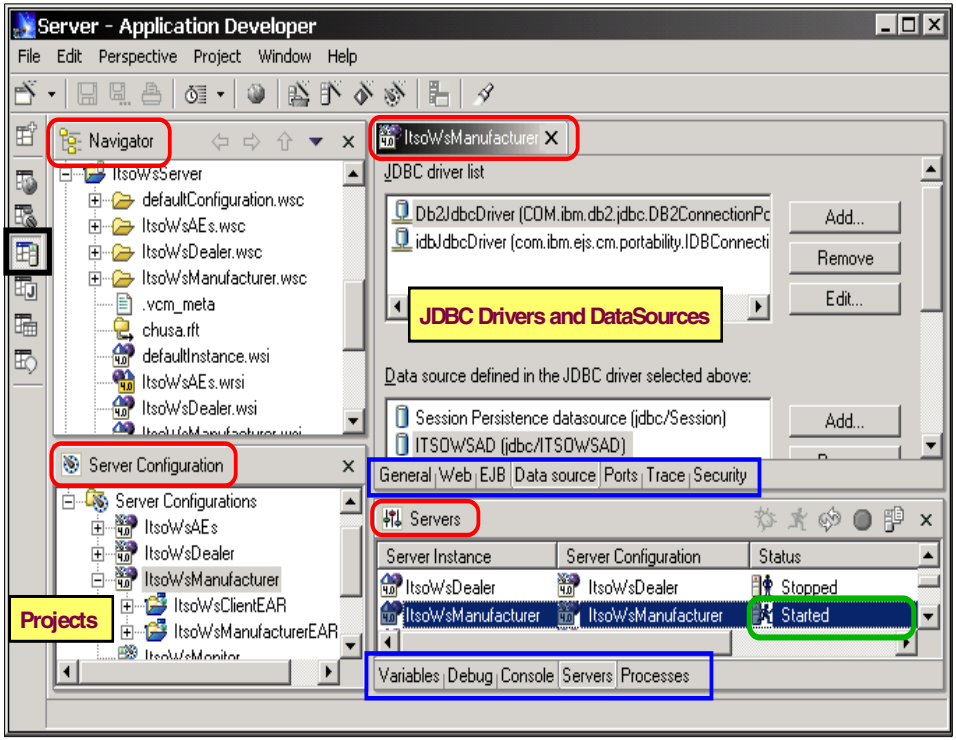


Figure 16-1 Server perspective

The Server perspective, Figure 16-1, contains four panes:

- **Top left**: Shows the Navigator, which displays the logical structure and resources of the project.
- **Top right**: Reserved for editors and browsers.

- **Bottom left**: Shows the Server Configuration view with all defined server instances and their corresponding configurations.
- **Bottom right**: Shows Server Control Panel - the controlling center for server starting, stopping, debugging and tracing.

16.3 Creating a server project automatically

To have Application Developer create a server instance and server configuration for you automatically, all you need to do is select *Run on Server* from the context menu while selecting the project (Figure 16-2).

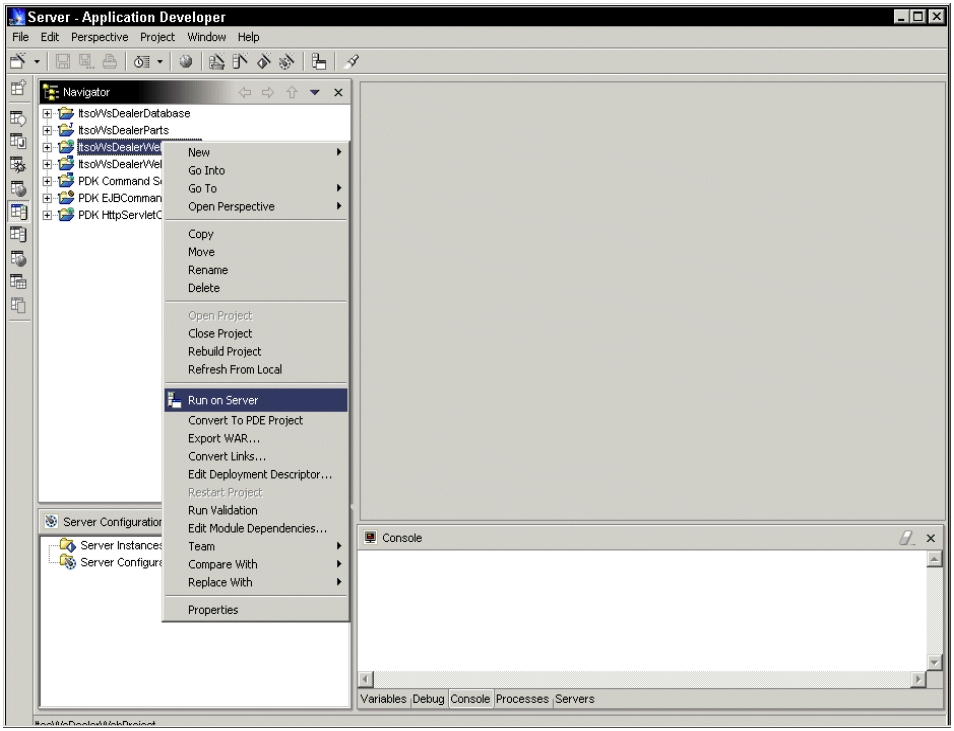


Figure 16-2 Select Run on Server from the context menu

Note: Before you run the application for the first time, notice that no server instances or server configurations exist.

From its context menu select the **Run on Server**. The Server Tools feature automatically does the following for you:

- **Launches** the Server Perspective window.
- **Creates** a server project with `Servers` as the default name.
- **Creates** the server instance with `WebSphere v4.0 Test Environment` as the default name.
- **Creates** the server configuration with `WebSphere Administrative Domain` as the default name.
- **Sets** the server instance to use the server configuration.
- **Adds** your project to the server configuration.
- **Starts** the server instance. (It may take certain amount of time to start it, depending on the server configuration settings.)
- **Opens** the Debug view and the Source pane, if there are breakpoints set in the file.
- **Displays** the file in the Web browser.

A server project contains two types of Server Tools resources: server instances and server configurations. The server project stores information about the test and the published servers and their configurations. If no server instances or server configurations exist, the first time you select *Run on Server* Application Developer will create for you Server project and the server instances and server configurations as seen in the server configuration view (Figure 16-3).

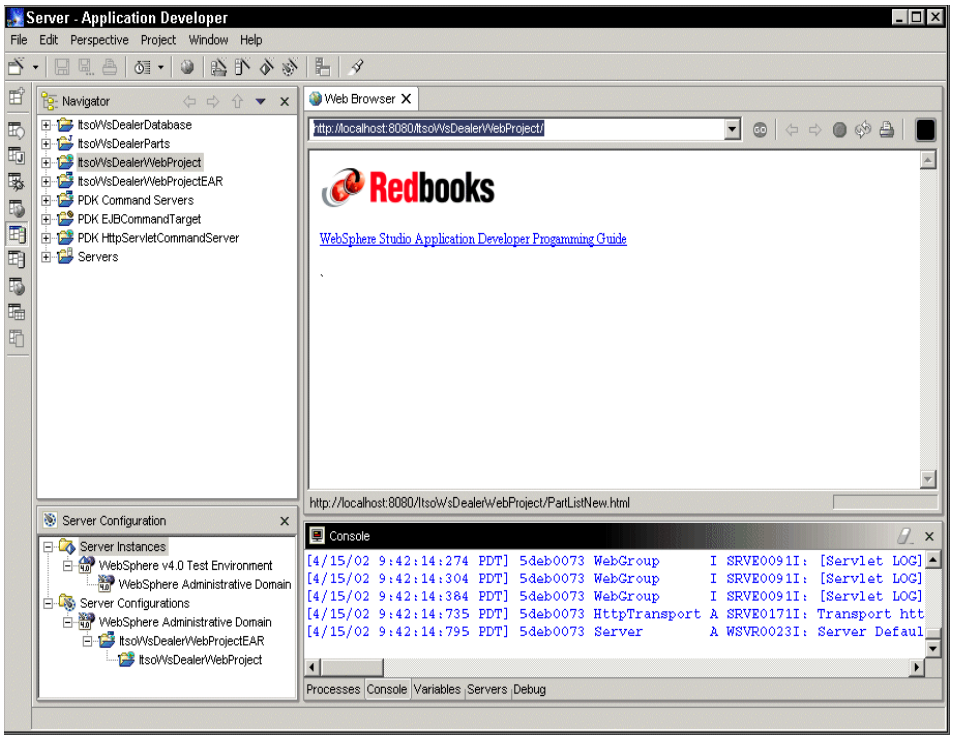


Figure 16-3 Automatic creation of server instances and server configurations

The server instance you get will be the WebSphere V4.0 Test Environment. This instance is the “for free” version of WebSphere Application Server AEs that ships with Application Developer. Once the server is started and running, you can proceed to test and debug your application.

Note: *Only if you use WebSphere Application Server:* If you want to publish your application remotely later, you may configure the server instance now - at the time of its creation.

16.4 Manual server instances and configurations

To create server instances and server configurations you can use and existing server project or create one of you own. Section 16.3, “Creating a server project automatically” on page 367 showed you how to let Application Developer create a local server instance and server configuration for you. This section will walk through the process of creating a new Server project and adding a remote server instance and server configuration for the ItsoWsDealerServerProject.

16.4.1 Creating a Server project

From the **File** menu, select **New—>Project**. The New Project wizard opens.

Select **Server** in the left frame and then select **Server Project** in the right frame (Figure 16-4).

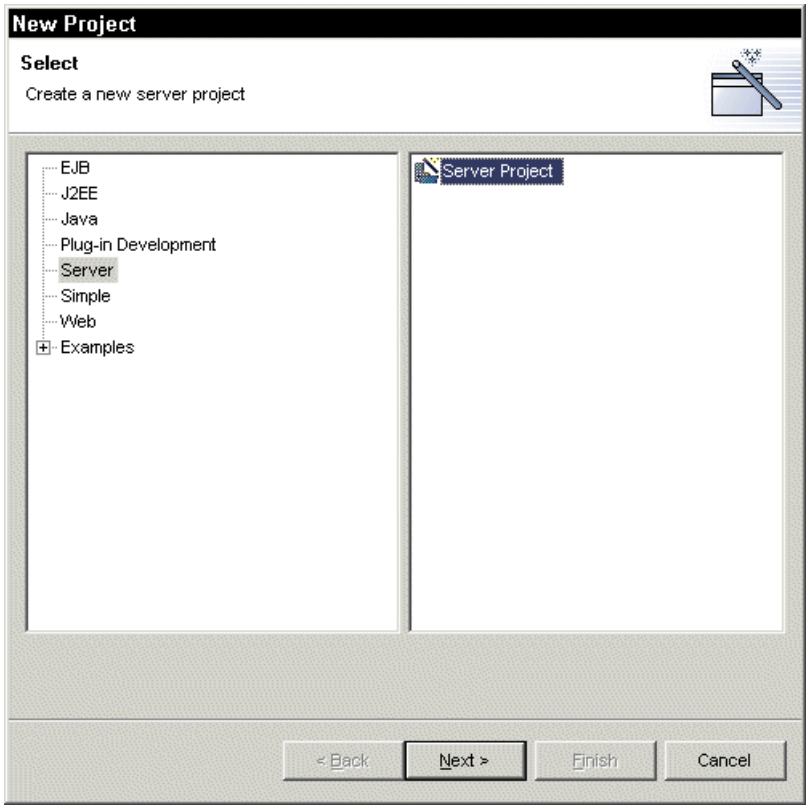


Figure 16-4 Creating a server project

Click **Next**. The Create a New Server Project wizard opens (Figure 16-5).

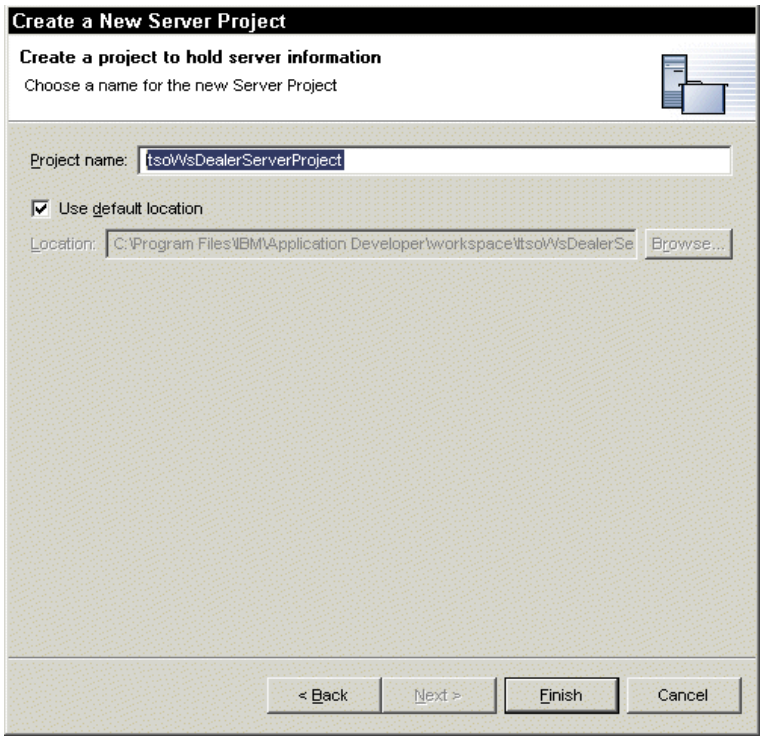


Figure 16-5 Create server project wizard

3. In the **Project name** field, type a name for the server project.

> **Note:** If you want to store the newly created server project in the default location, select the **Use Default Location** check box. However, if you want to store the newly created server project in another location, either type the location in the **Location** field or click **Browse** to select it.

Click **Finish**. A new server project appears in the Navigator view of the Server perspective.

16.5 Creating an instance and configuration separately

With Application Developer you can create the server instance and configuration with a combined wizard or you can create them with independent wizards. You may want to create a server instance using an existing server configuration, create a server configuration using an existing server instance, or create the server instance and server configuration at the same time. Application Developer has a wizard for any option you chose.

16.5.1 Creating a server instance manually

Once you have created a server project, you can create a server instance to identify the run-time environment that you want to use for testing your project resources.

We will now show you how to manually create a server instance.

In the Navigator view, right-click the server project and then select **File—>New—>Other**. The New Select wizard opens.

Select **Server** in the left frame and then select **Server Instance** in the right frame (Figure 16-6).

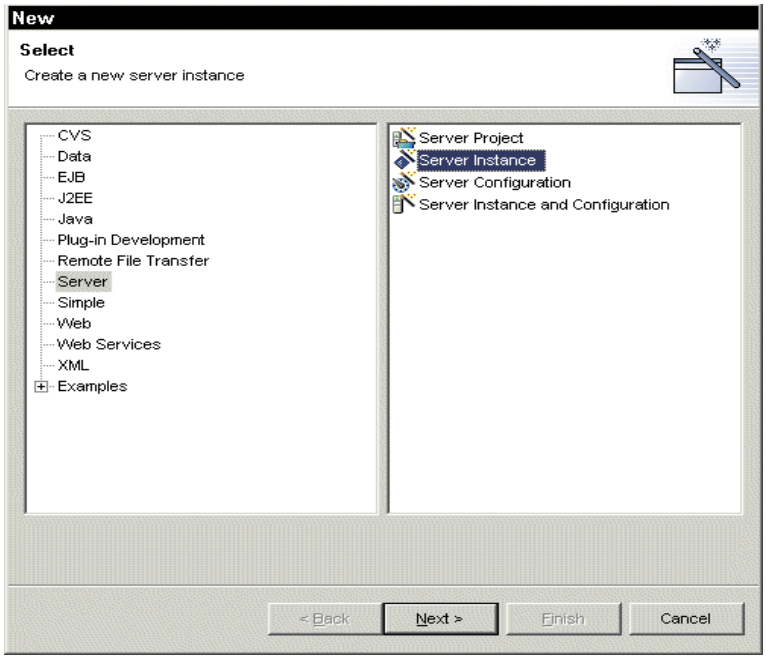


Figure 16-6 Creating a server instance

Click **Next**.

The Create a New Server Instance wizard opens (Figure 16-7). This wizard creates a new server instance, which contains information required to point to a specific run-time environment for local or remote testing, for publishing to an application server. For a remote server instance, there are two types of file transfer options you can choose. Both `copy` and `FTP` file transfer types will be shown when creating the `Websphere v4.0 Remote Server` instance in this example.

Note: For more information about any of the fields on this wizard, select the field and then press F1. This is true for all Application Developer dialogs.

Prerequisites: If you want to test your projects remotely on WebSphere Application Server, you must install the following applications on the remote server:

- IBM WebSphere Application Server Advanced Single Server Edition for Multi platforms
- IBM Agent Controller
- (Optional) FTP server

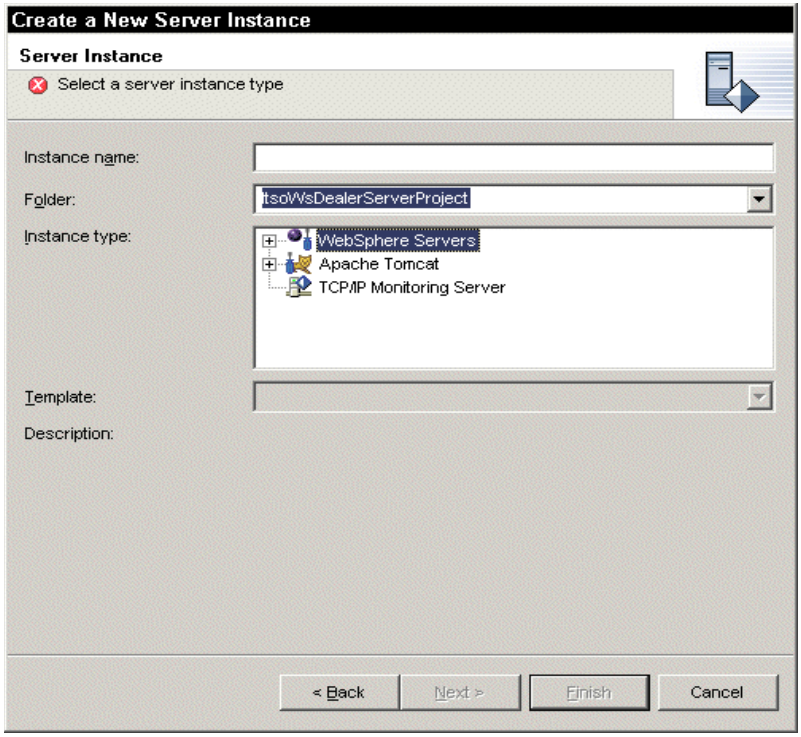


Figure 16-7 Create remote instance wizard

In the **Instance Name** field, enter a name for the server instance. On the next few screens you will see two methods, copy or ftp, for the remote file transfer. It is a good idea to name the instance something meaningful so for now just enter the name *ItsoWsDealerInstanceCopy* for copy or *ItsoWsDealerInstanceFTP* for FTP which will be explained later (Figure 16-8).

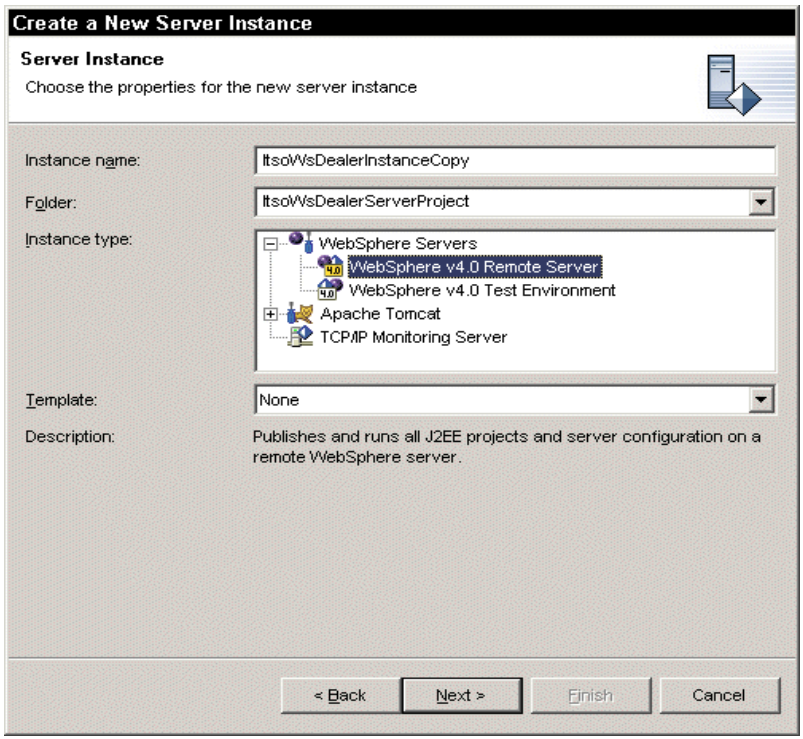


Figure 16-8 Create remote copy instance

Important: Make sure you select the new server project on the Folder field of the wizard.

If you want to create a server instance based on an existing template, select the name of the template from the **Template** list. Otherwise, select **None**.

Depending on what instance type you select, the **Next** button may be enabled therefore allowing you to specify the details of the server instance on the additional page(s) of the Create a New Server Instance wizard. Click **Next**.

The WebSphere Remote Server Instance Settings opens allowing you to provide additional remote server instance information required for using the WebSphere Application Server remotely (Figure 16-9).

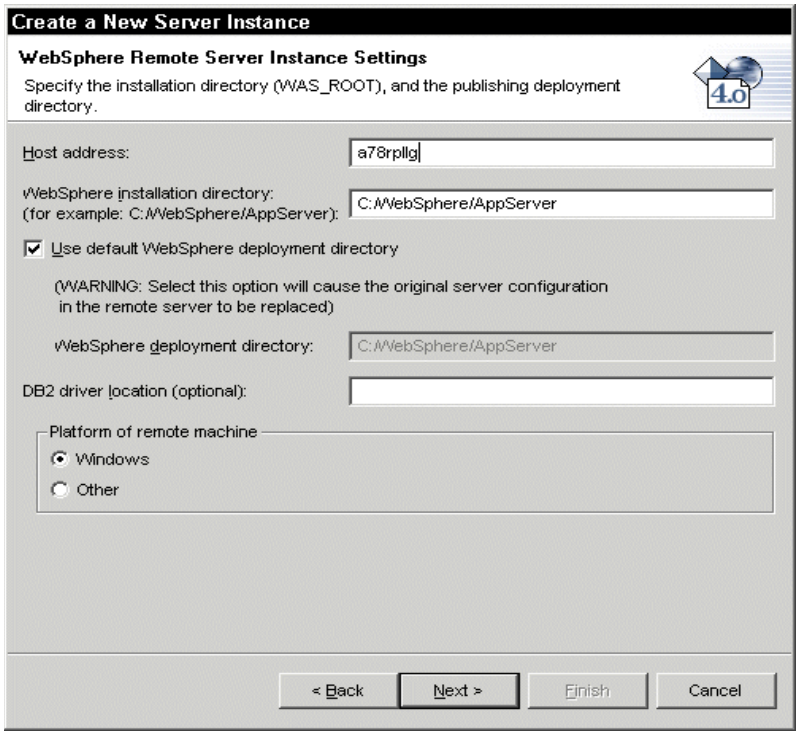


Figure 16-9 Create remote server instance settings

In the **Host address** field, type the fully qualified DNS name or the IP address of the remote machine that WebSphere Application Server is running on. The field is pre-filled with the default address for the local host (127.0.0.1).

Note: All paths on this page are seen from the remote machine.

In the **WebSphere installation directory** field, type the path where you installed WebSphere Application Server on the remote machine. This path is the same as the WAS_ROOT path mappings as defined by the WebSphere server configuration.

If you select the **Use default WebSphere deployment directory** check box, then you want to use the default WebSphere deployment directory. The **WebSphere deployment directory** field is then pre-filled with the default value. Otherwise, in the **WebSphere deployment directory** field, type the path of the directory where the Web application and server configurations will be published.

This directory is any existing directory that can be seen from the remote server machine. For example, if the directory E:/testdir resides on the remote machine, then type E:/testdir in this field. However, if you are following WebSphere naming conventions and install WebSphere Application Server in the C:/WebSphere/AppServer directory, then the WebSphere deployment directory is C:/WebSphere/AppServer.When publishing to the remote server, the server configuration and the Web application will be published to a directory under the remote deployment directory called "config" and "installedApps" respectively.

Notes:

- If you select the *Use default WebSphere deployment directory* check box when creating a remote server instance and then publish using this instance, the default WebSphere Application Server `server-cfg.xml` file is replaced with the published version.
- If you select the *Use default WebSphere deployment directory* check box when creating a remote server instance and you also select the *Generate plug-in configuration file* check box when editing a remote server instance and then publish using this instance, the default WebSphere Application Server `server-cfg.xml` file and `plugin-cfg.xml` files are replaced with the published version.

Optional: In the **DB2 driver location** field, type the DB2 location where the DB2 classes reside in the remote machine. If the default value is set in the Preference - WebSphere page, this field is pre-filled with the DB2 location.

Click **Next** again to display the third page of the server creation wizard. This page allows you to define a remote file transfer instance. A *remote file transfer instance* contains information for transferring Web applications and server configurations to the remote server during publishing (Figure 16-10).

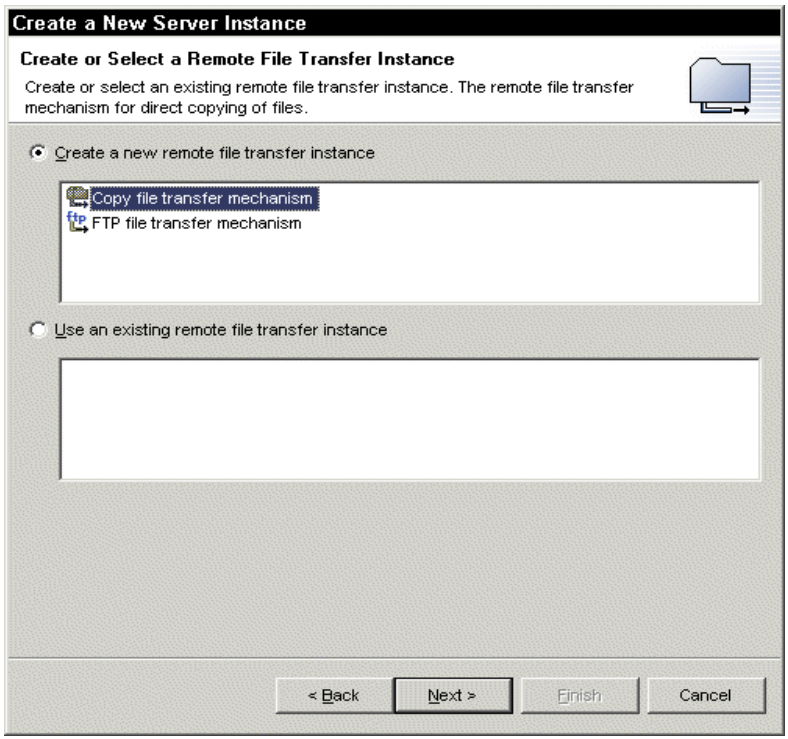

Create a New Server Instance
Create or Select a Remote File Transfer Instance
Create or select an existing remote file transfer instance. The remote file transfer mechanism for direct copying of files.
Create a new remote file transfer instance
Copy file transfer mechanism
FTP file transfer mechanism
Use an existing remote file transfer instance
< Back
Next >
Finish
Cancel

Figure 16-10 Select new or existing remote file transfer

If the **Create a new remote file transfer instance** radio button is selected:

From the list, select one of the following:

- **Copy file transfer mechanism**, to copy resources directly from one machine to another in the file system.
- **FTP file transfer mechanism**, to copy resources from one machine to another using File Transfer Protocol (FTP).

Note: If the *Use an existing remote file transfer instance* radio button is selected, the Next button is not enabled. Select the remote file transfer instance that you want to use to transfer files remotely and omit the next step.

There are two types of remote file transfers in Application Developer, Copy or FTP. Follow the next step to use the Copy file transfer mechanism. Skip the next step if **FTP file transfer mechanism** is selected.

If **Copy file transfer mechanism** is selected, the next page of the wizard appears (Figure 16-11).

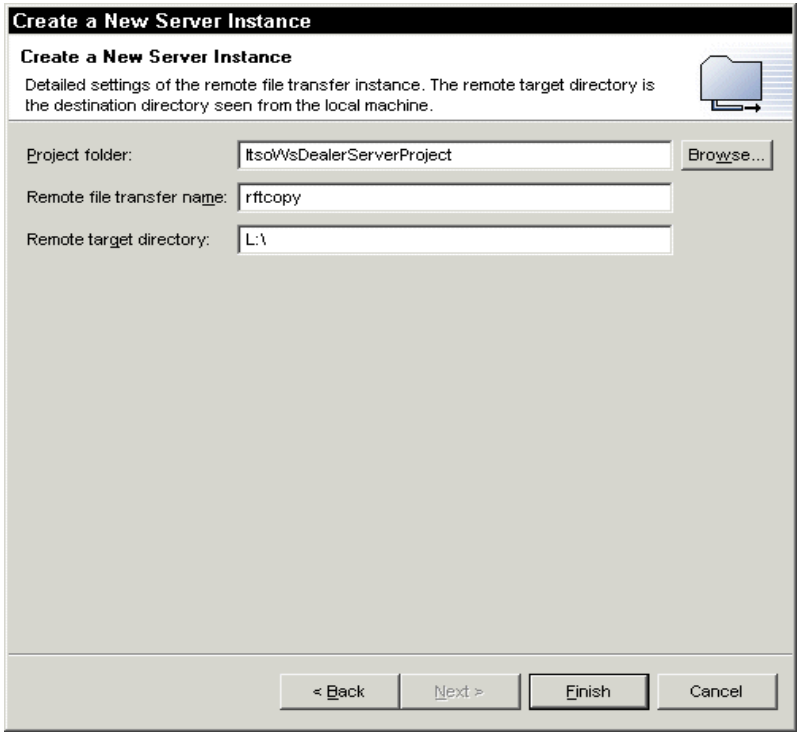


Figure 16-11 Create remote copy transfer instance

- In the **Project folder** field, type the name of the project folder where the remote file transfer instance will reside
- In the **Remote file transfer name** field, type name of the remote file transfer instance. Our example the name is `rftcopy`
- In the **Remote target directory** field, type the remote target directory where you want your applications and server configuration published. This remote target directory is the one as seen by the local machine. If WebSphere Application Server is installed on a different machine, then the remote target directory is the network drive that maps to the WebSphere deployment directory. If WebSphere Application Server is installed on the same machine as the workbench, then the remote target directory should be the same as the contents in the **WebSphere deployment directory** field (`for example, C:/WebSphere/AppServer`).
- Click **Finish** and skip the next step.

If **FTP file transfer mechanism** is selected, the next page of the wizard appears (Figure 16-12).

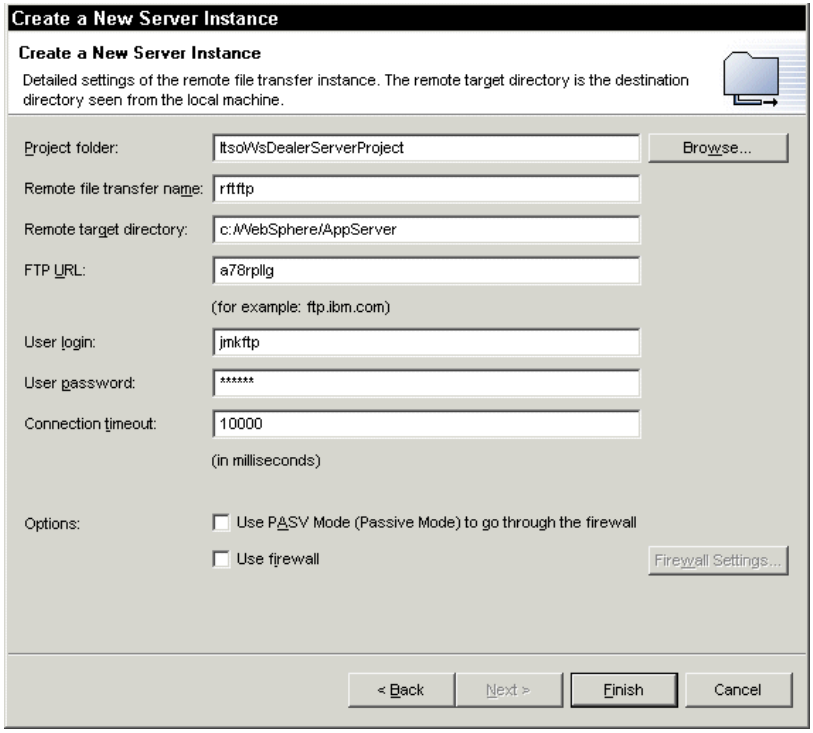


Figure 16-12 Create remote FTP server instance

- In the **Project folder** field, type the name of the project folder where the remote file transfer instance will reside.
- In the **Remote file transfer name** field, type the name of the remote file transfer instance. Our example the name is `rftftp`
- In the **Remote target directory** field, type the remote target directory where you want your application and server configuration published. This remote target directory points to the WebSphere deployment directory that is seen from the workbench using the FTP client program. For example, if the WebSphere deployment directory is C:/WebSphere/AppServer and your FTP server route directory is C:/, then your remote target directory is /WebSphere/AppServer.

Note: To determine whether a beginning slash is required, log on to the FTP server using a FTP client program, and then type the pwd command. If the results containing the default log on directory begins with a slash, then a slash is required prior to typing the remote target directory in the **Remote target directory** field.

- In the **FTP URL** field, type the URL that is used to access the FTP server.
- In the **User login** field and the **User password** field, type the FTP user ID and password that will be used to access the FTP server.
- In the **Connection time-out** field, type the time (in milliseconds) that the workbench will attempt to contact the FTP server before timing out.
- Select the **Use PASV Mode (Passive Mode) to go through the firewall** check box, if you want to pass through a firewall — provided that one is installed between your FTP server and the workbench.
- Select the **Use Firewall** check box, if you want to use the firewall options.
- To change the firewall options, select the **Use Firewall** check box, and then click **Firewall Settings**.
- Click **Finish**

The instances created will reside locally on your machine. The new server instances appear in project folder the Server Configuration view, in the Server Configuration view under the **Server Instances** folder, and in the Servers view under the **Server Instance** column. The remote file transfer instances appear in the Navigator view (Figure 16-13).

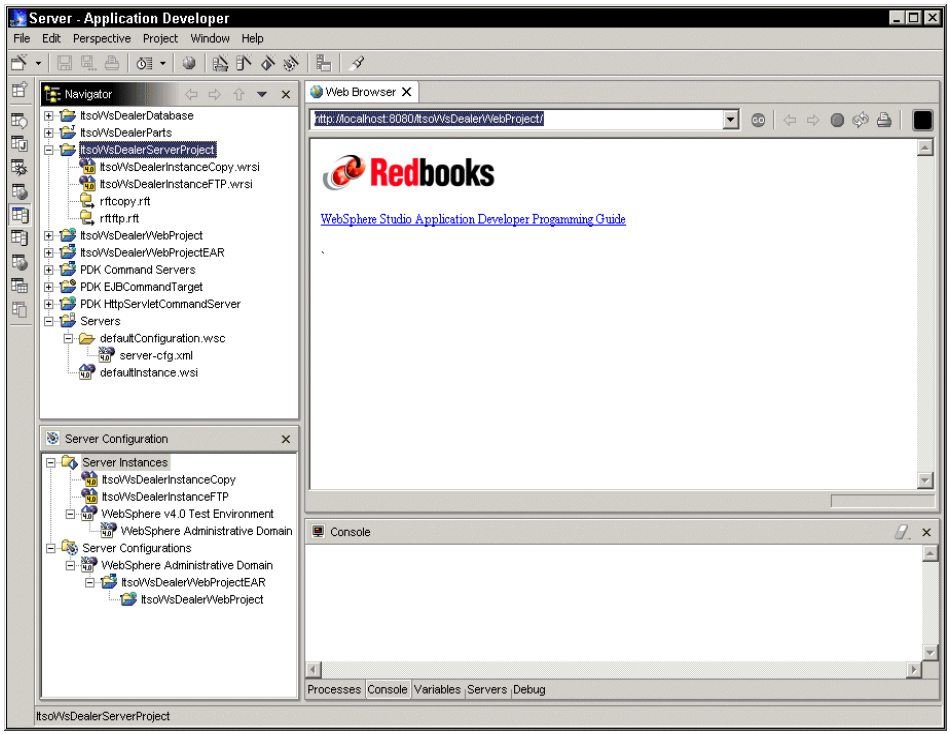


Figure 16-13 Remote server instances appear

16.5.2 Creating a server configuration manually

Once you have created a server instance, you can create a server configuration. This server configuration contains information that is required to set up and publish a server. Later you will add the server configuration to the server instance.

To manually create a server configuration:

In the Navigator view, right-click the server project and then select **File—>New—>Other.** The New wizard opens (Figure 16-14).

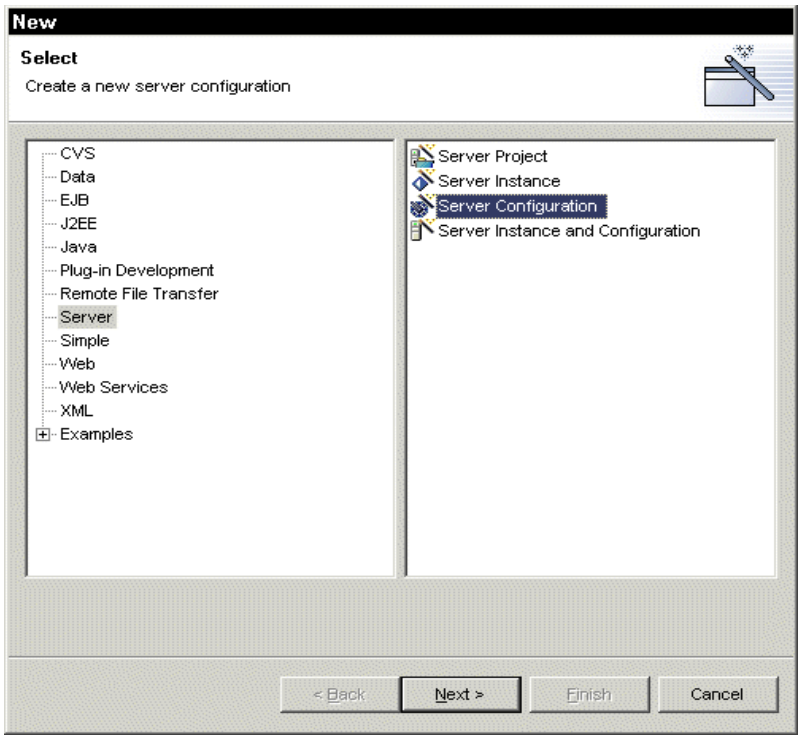


Figure 16-14 Creating a server configuration

Select **Server** in the left frame and then select **Server Configuration** in the right frame.

Click **Next**. The Create a New Server Configuration wizard opens (Figure 16-15). This wizard creates a new server configuration which contains information required to set up a server.

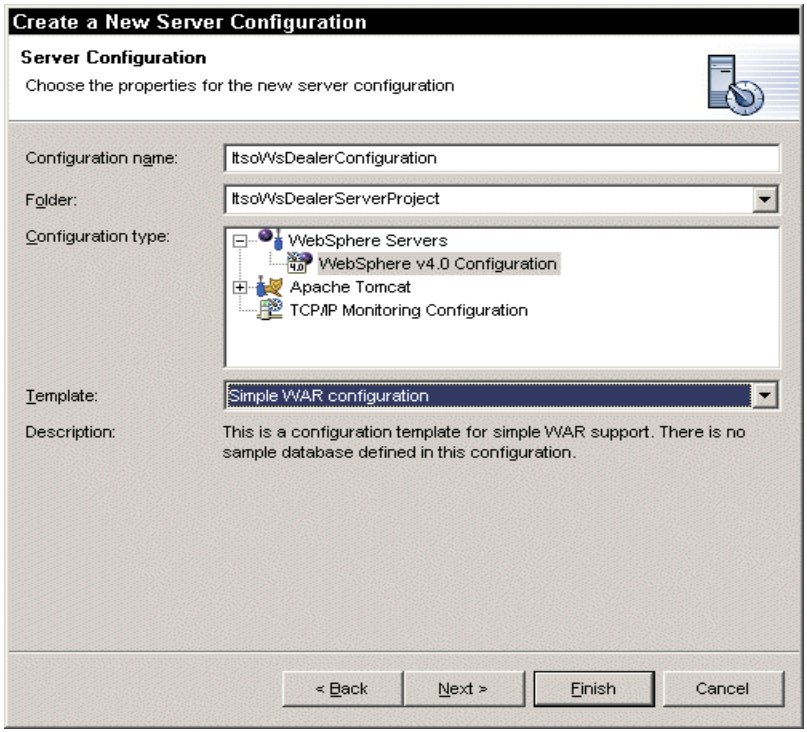


Figure 16-15 Create WebSphere configuration

In the **Configuration type** list box, select the type of server that this configuration will use to set up and publish the server.

Note: Depending on what configuration type you select, the Next button may be enabled therefore allowing you to specify the details of the server configuration. For example, on the second page of the Create a New Server Configuration wizard, you may want to change the default HTTP port number (8080) for the WebSphere Application Server.

If you want to create this server configuration based on an existing template, select the name of the template from the **Template** list box. Otherwise, select **None**. **For WebSphere Application Server:** Select **Simple WAR configuration** if you want to base the server configuration on a template that is optimized for WAR files.

Click **Next** if you want to change the HTTP port number (Figure 16-16).

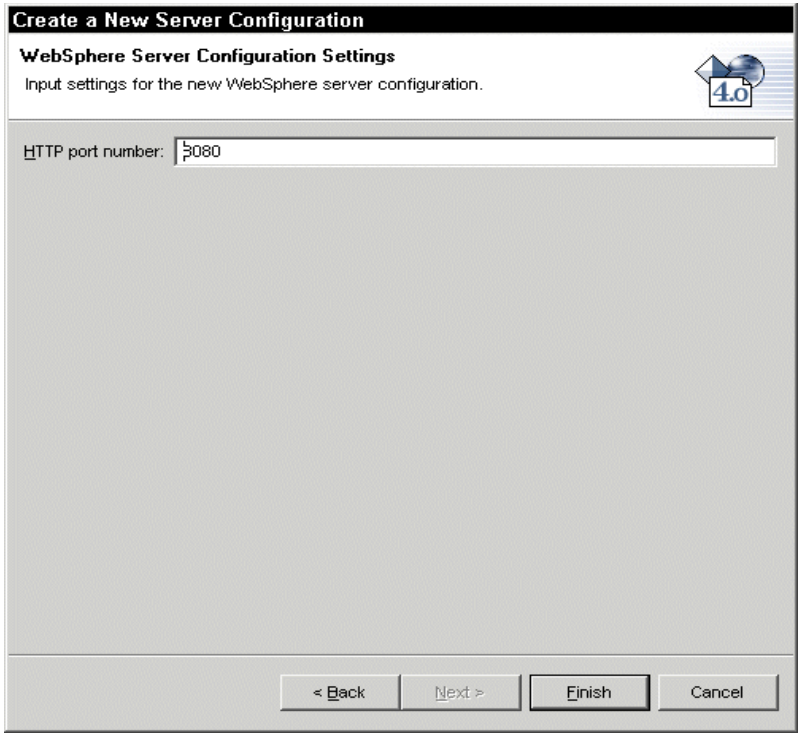


Figure 16-16 Change HTTP port number

Click **Finish**. The new server configuration folder is created under the server project folder in the Navigator view and in the Server Configuration view under the **Server Configurations** folder (Figure 16-17).

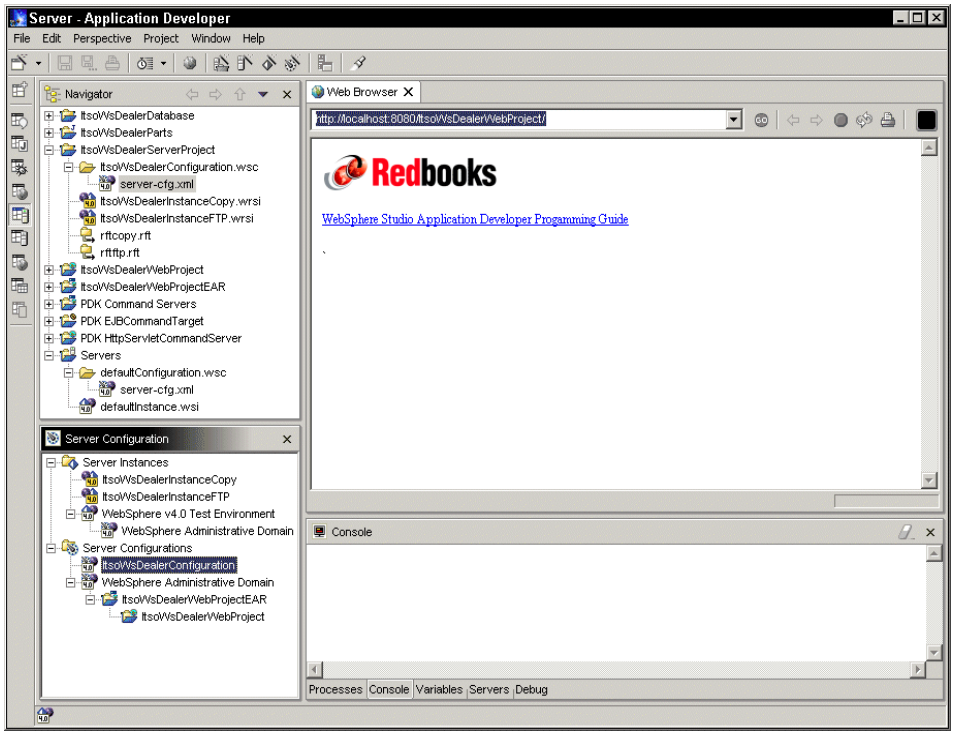


Figure 16-17 Server configuration appears

16.6 Creating an instance and configuration together

You can create both server instance and configuration at once by doing the following:

In the Navigator view, right-click the server project and then select **New—>Server Instance and Configuration**. The Create a New Server Instance and Server Configuration wizard opens (Figure 16-18). This wizard creates a new server instance and server configuration at the same time. A server instance contains information required to point to a specific run-time environment for local or remote testing, or for publishing to an application server. A server configuration contains information required to set up and publish to a server.

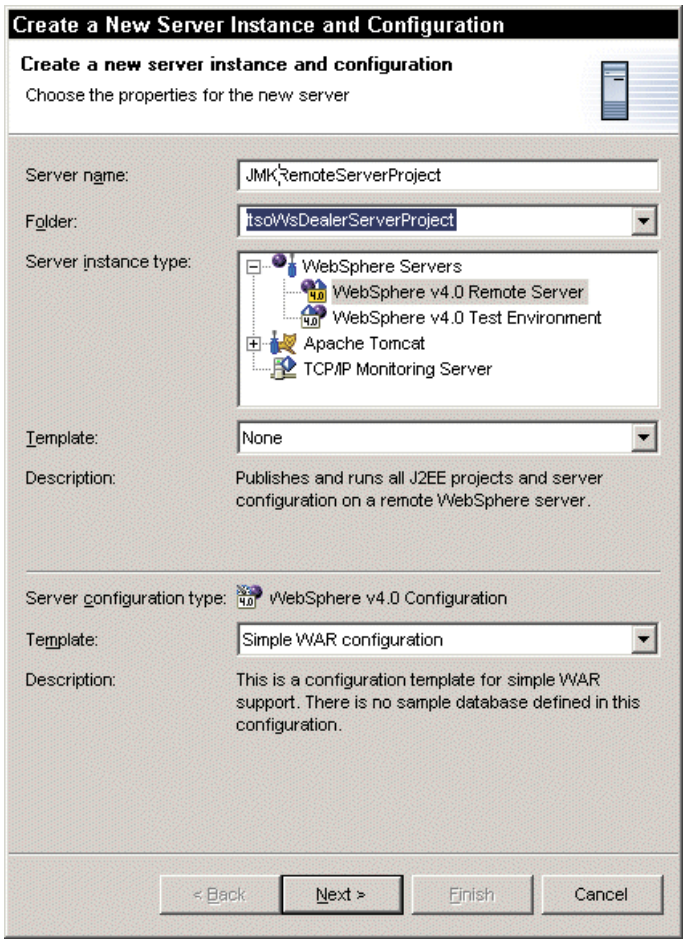


Figure 16-18 Combined instance and configuration wizard

In the Server instance type list, select the type of test environment where you want to test your resources. For more details on each of the supported instance types, select a type from the list and read the description that appears in the Description area of this wizard page.

If you want to create this server instance based on an existing one, select the name of the template from the top **Template** list. Otherwise, select **None**.

Depending on what server instance type you select, the server configuration type is automatically selected for you. If you want to create this server configuration based on an existing one, select the name of the template from the lower **Template** list. Otherwise, select **None**. For WebSphere Server instances select the **Simple WAR Configuration** as the type.

Depending on what server instance type you select, the **Next** button may be enabled therefore allowing you to specify the details of the server on the additional page(s) of the creating wizard. If you want to test remotely, you can specify the remote details on the additional page(s). For details on creating the server instance or server configuration, see 16.5.1, “Creating a server instance manually” on page 372 and 16.5.2, “Creating a server configuration manually” on page 382.

Click **Finish**. The new server instance and the new server configuration folder appear under the server project folder in the Navigator view.

16.7 Adding a project to a server configuration

Once you have set a server configuration to a server instance, you need to create a Server Tools relationship between your project and your server configuration. If you are using WebSphere Application Server, this relationship is between your Enterprise Application project and your server configuration. If you are using Tomcat, this relationship is between your Web project and your server configuration.

This is done by adding your project to a server configuration.

To open the Server Configuration view, on the main menu bar, click **Perspective—>Show View—>Server Configuration.**

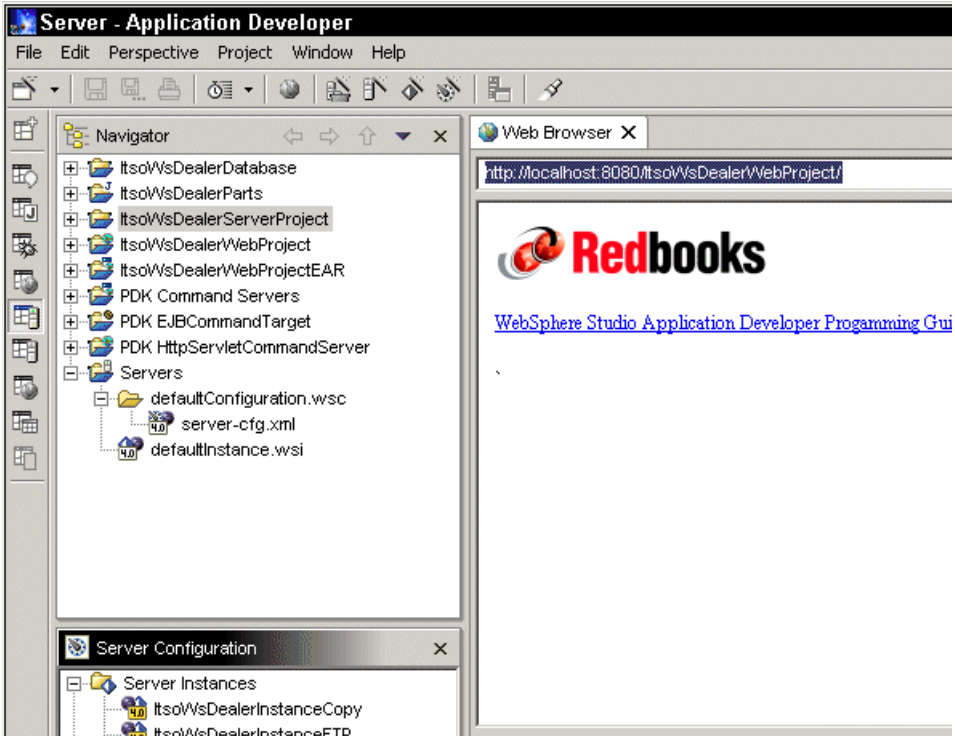


Figure 16-19 Add project to configuration

Expand the **Server Configurations** folder.

Right-click on the appropriate configuration, and select **Add project**.

Select the appropriate project that you want to test with the server configuration. The project name appears under the server configuration in the Server Configuration view.

Note: You can only select one project at a time.

16.8 Replacement of default files

If you select the **Use default WebSphere deployment directory** check box when creating a remote server instance and then publish using this instance, the default WebSphere Application Server `server-cfg.xml` file is replaced with the published version.

If you select the **Use default WebSphere deployment directory** check box when creating a remote server instance and you also select the **Generate plug-in configuration file** check box when editing a remote server instance and then publish using this instance, the default WebSphere Application Server `server-cfg.xml` file and `plugin-cfg.xml` files are replaced with the published version.

Before `server-cfg.xml` and `plugin-cfg.xml` files are replaced, backup copies of these are saved in the *WebSphere_deployment_directory*/config/wasTools_bkup directory (where *WebSphere_deployment_directory* is the path of the directory where the Web application and server configurations are published). Backup copies are only made the first time in the day that you publish remotely. The file name of the backup files has the current date appended to it. For example, server-cfg_*YYYYMMDD*.xml (where *YYYY* is the year, *MM* is the month, and *DD* is the day).

To use a backup copy of a file:

- Rename the file removing the date.
- Copy to the *WebSphere_deployment_directory*/config directory.

16.9 Starting a remote instance

To start a remote instance using the Server Tools feature on WebSphere Application Server, follow these steps:

Install the *WebSphere Application Server AEs for Multiplatforms* on the remote machine where you want to test your project.

Important: We recommend that you install the WebSphere Application Server before you install the IBM Agent Controller. Otherwise, you need to follow the instructions below to manually change the settings in the server configuration.

Install the *IBM Agent Controller* on the remote machine where you want to test your project by following the instructions in the installation guide for this product. For more information on how to use the Agent Controller, refer to the Agent Controller online documentation.

Note: If the Agent Controller was installed before WebSphere Application Server was installed, then you must follow the instructions to edit the configurations described in the installation guide for this product.

Start the Agent Controller on the remote machine. For more information on how to start and stop the Agent Controller, refer to the Agent Controller online documentation.

Note: If you are using an FTP server, first ensure that it is started.

Create a **Server Instance and Configuration** to run remotely. See Section 16.4, “Manual server instances and configurations” on page 369.

Optional: To test using the IBM HTTP server in the remote server, select the Generate plug-in configuration check box on the General page of the remote server instance editor. The plugin-cfg.xml file will be generated and published under the config directory, under the WebSphere remote deployment directory.

By default, when the remote server starts, the system classpath for the remote machine is added to the remote server process. If you do not want to add the system classpath to the remote server process, then do the following before you start the remote server:

If the remote server instance is started, then stop the server instance.

- Edit the serviceconfig.xml file located in the *config* directory under the Agent Controller on the remote machine.
- Change the *position* parameter of the CLASSPATH variable for the wteRemote.exe application from "prepend" to "replace".
- Save your changes.
- Restart the Agent Controller.
- Start the server instance.

Note: If you start the WebSphere remote server instance without starting Agent Controller first, you will receive an error message.

When the server is started, the Server Tools feature automatically does the following for you:

- Publishes the server configuration on the remote machine.
- Publishes the application that you want to test on the remote machine.
- Starts the server.

Note: Do not stop the Agent Controller when a remote server is starting or started; otherwise, the server process in the remote server may not be properly stopped. You then may need to manually terminate the remote server process in the remote machine using the Windows Task Manager

16.10 Stopping remote instance

Note: Do not stop the Agent Controller when a remote server is starting or started; otherwise, the server process in the remote server may not be properly stopped. You then may need to manually terminate the remote server process in the remote machine using the Windows Task Manager.

When you have finished testing your project remotely, follow these steps:

- Stop the server instance.
- Stop the IBM Agent Controller on the remote machine.

16.11 Apache Tomcat

Follow the steps described in 16.5, “Creating an instance and configuration separately” on page 372, for local server instance, but select Tomcat as the type.

Note: Apache Tomcat can only be configured to run in a local configuration.

17

Testing and debugging your application

This chapter describes the following:

- Debug perspective
- Debugging a Java program in WebSphere test environment
- Setting breakpoints
- Debug functions
- Watching variables
- Debugging on a remote server

© Copyright IBM Corp. 2002

17.1 Debug perspective

See 3.10, “Debug perspective” on page 91 for a description of this perspective.

When Debugging JSPs, make sure the **Generate debug information when compiling JSPs** check box is checked. From the Server perspective locate the Server Configuration pane and double-click the server instance to bring up the window in Figure 17-1.

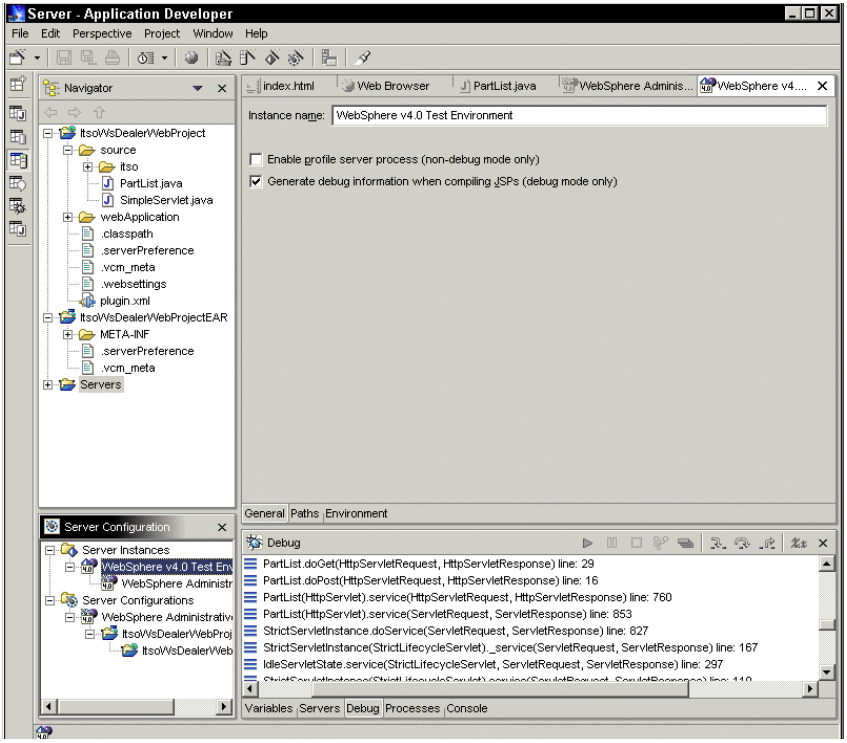


Figure 17-1 Server instance parameters

17.2 Debugging a Java program

In this section, we will show how to debug a Java program running as part of a Web application. We will be using *PartList.java* for our example. This is the servlet that was developed in Chapter 8, “Creating Web applications with dynamic content” on page 203.

17.2.1 Setting breakpoints in the code

To add a breakpoint in the code, do the following:

1. In the Packages view in the Java perspective, double-click **PartList.java** to open it in an editor.
2. Place your cursor in the gray bar (along the left edge of the editor area) on the line `PartListBean = new PartListBean().`
3. Double-click to set a breakpoint (Figure 17-2). A marker will be shown at the line.



Figure 17-2 Adding a breakpoint

Note: The breakpoint is initially shown as blue because it is unverified, that is the containing class has not yet been loaded by the Java VM.

17.2.2 Testing the application with breakpoints enabled

Once you have set the breakpoint, the Web application can be started. Follow these steps to start your debug session:

From the Server Perspective, bring up the context menu on ItsoWsDealerWebProject and select **Run on Server** (Figure 17-3).

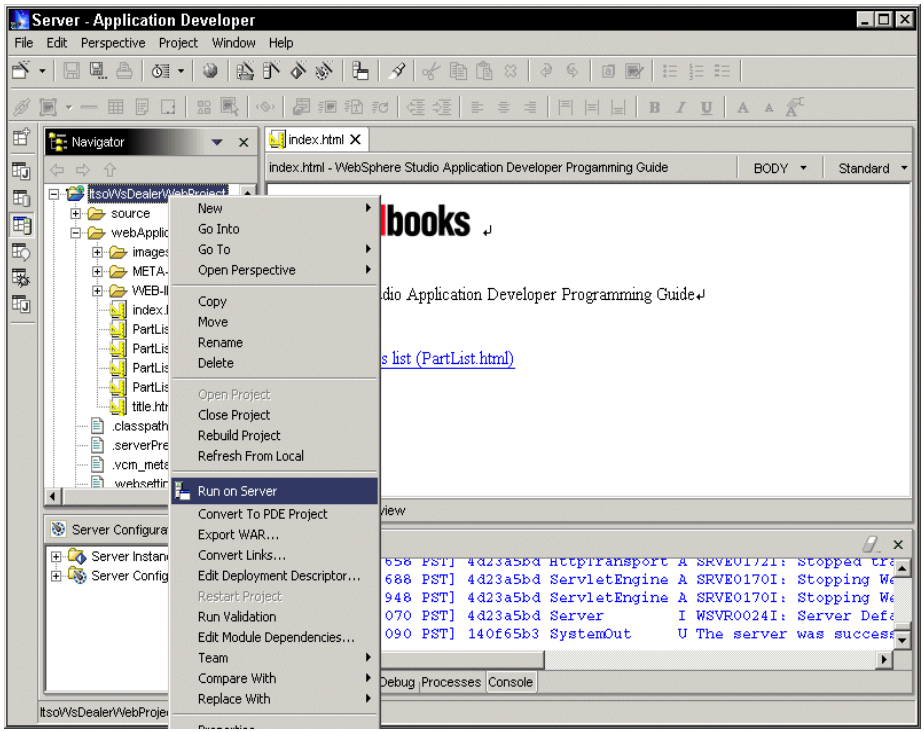


Figure 17-3 Run Web project on server

In the Debug Perspective you should now see `index.html` displayed in the Web Browser View as in Figure 17-4.

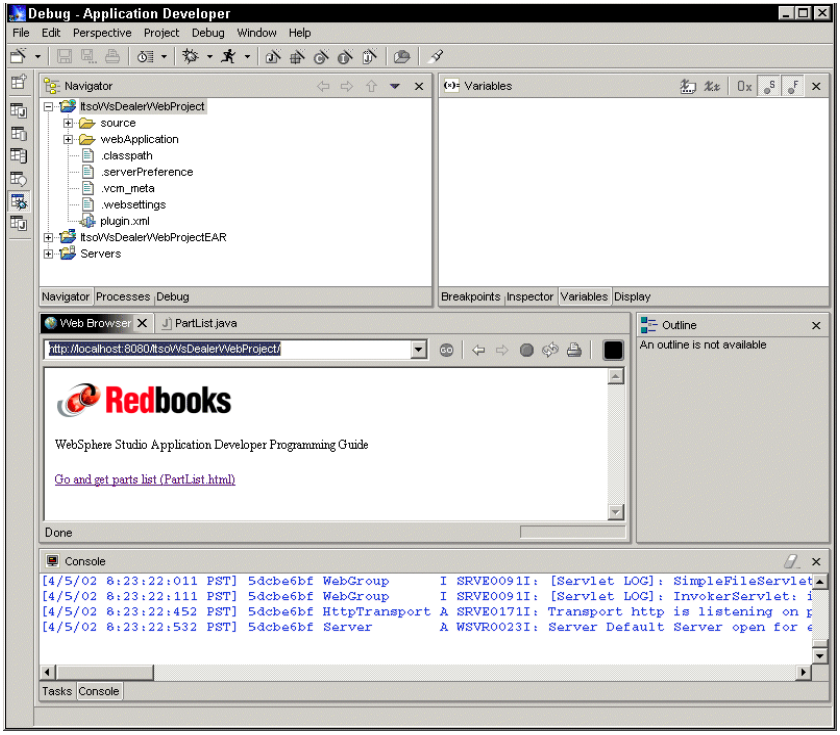


Figure 17-4 Index.html displayed in Web browser

Click on the **Go and get parts list** link on index.html to display the PartList.html form page.

Enter a search argument, "**RR**", on the form and click **Retrieve** (Figure 17-5).

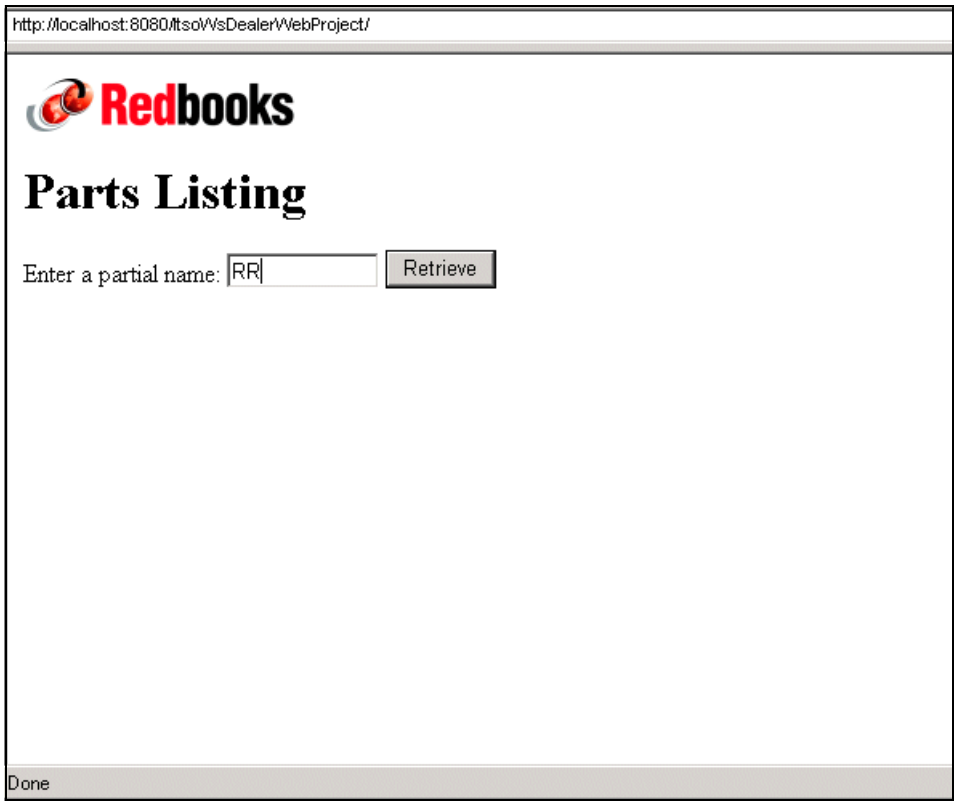


Figure 17-5 Submit a query

As soon as the breakpoint in `PartList.java` is hit, execution will stop and the `PartList.java` source file will be displayed with the line containing the break point highlighted. Notice that the process is still active (not terminated) in the Processes view. Other threads might also still be running (Figure 17-6).

Note: If the Debug Perspective is not displayed automatically when the breakpoint is reached, you can open it from the **Open Perspective** menu item.

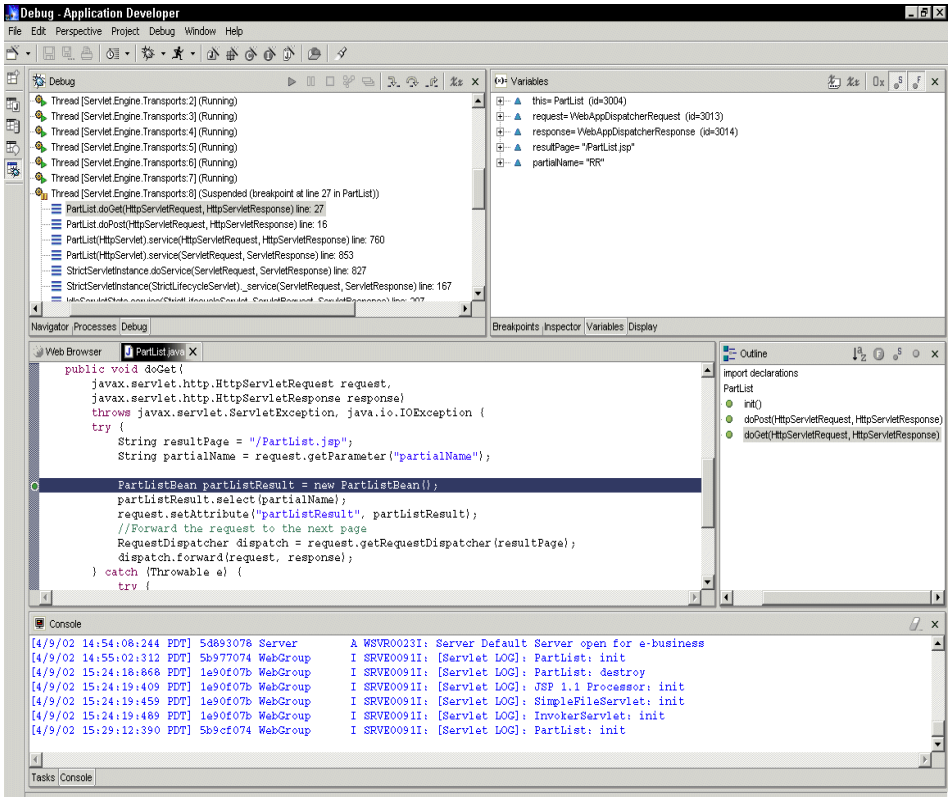

Debug - Application Developer
File Edit Perspective Project Debug Window Help
Debug
Thread [Servlet Engine.Transports:2] (Running)
Thread [Servlet Engine.Transports:3] (Running)
Thread [Servlet Engine.Transports:4] (Running)
Thread [Servlet Engine.Transports:5] (Running)
Thread [Servlet Engine.Transports:6] (Running)
Thread [Servlet Engine.Transports:7] (Running)
Thread [Servlet Engine.Transports:8] (Suspended (breakpoint at line 27 in PartList))
PartList.doGet(HttpServletRequest, HttpServletResponse) line: 27
PartList.doPost(HttpServletRequest, HttpServletResponse) line: 16
PartList(HttpServlet).service(HttpServletRequest, HttpServletResponse) line: 760
PartList(HttpServlet).service(ServletRequest, ServletResponse) line: 853
StrictServletInstance.doService(ServletRequest, ServletResponse) line: 827
StrictServletInstance(StrictLifecycleServlet)._service(ServletRequest, ServletResponse) line: 167
Navigator Processes Debug
Variables
this= PartList (id=3004)
request= WebAppDispatcherRequest (id=3013)
response= WebAppDispatcherResponse (id=3014)
resultPage= "/PartList.jsp"
partialName= "RR"
Breakpoints Inspector Variables Display
Web Browser
PartList.java
public void doGet(
javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException {
try {
String resultPage = "/PartList.jsp";
String partialName = request.getParameter("partialName");
PartListBean partListResult = new PartListBean();
partListResult.select(partialName);
request.setAttribute("partListResult", partListResult);
//Forward the request to the next page
RequestDispatcher dispatch = request.getRequestDispatcher(resultPage);
dispatch.forward(request, response);
} catch (Throwable e) {
try {
Outline
import declarations
PartList
init()
doPost(HttpServletRequest, HttpServletResponse)
doGet(HttpServletRequest, HttpServletResponse)
Console
[4/9/02 14:54:08:244 PDT] 5d893078 Server A WSVR0023I: Server Default Server open for e-business
[4/9/02 14:55:02:312 PDT] 5b977074 WebGroup I SRVE0091I: [Servlet LOG]: PartList: init
[4/9/02 15:24:18:868 PDT] 1e90f07b WebGroup I SRVE0091I: [Servlet LOG]: PartList: destroy
[4/9/02 15:24:19:409 PDT] 1e90f07b WebGroup I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
[4/9/02 15:24:19:459 PDT] 1e90f07b WebGroup I SRVE0091I: [Servlet LOG]: SimplePileServlet: init
[4/9/02 15:24:19:489 PDT] 1e90f07b WebGroup I SRVE0091I: [Servlet LOG]: InvokerServlet: init
[4/9/02 15:29:12:390 PDT] 5b9cf074 WebGroup I SRVE0091I: [Servlet LOG]: PartList: init
Tasks Console

Figure 17-6 Stop at breakpoint

Note: The breakpoint is green because is it now verified.

17.2.3 Debug functions

From the Debug view, which should now be displayed in the top left pane, you can use the functions available from its icon bar to control the execution of the application. The following functions are available:

- **Resume**: Runs the application to completion.
- **Terminate**: Terminates a process.
- **Suspend**: Suspends a running thread.
- **Step Into**: Steps into the highlighted statement.
- **Step Over**: Steps over the highlighted statement.
- **Run to Return**: Steps out of the current method.

In the upper right pane you will see the various debugging views that are available (Figure 17-7). You can use the *Breakpoint* view to display and manipulate the breakpoints that are currently set. In the *Inspector* and *Display* views you can see the result of inspecting and displaying parts of your code (you can access these functions from the context menu in the source code pane).

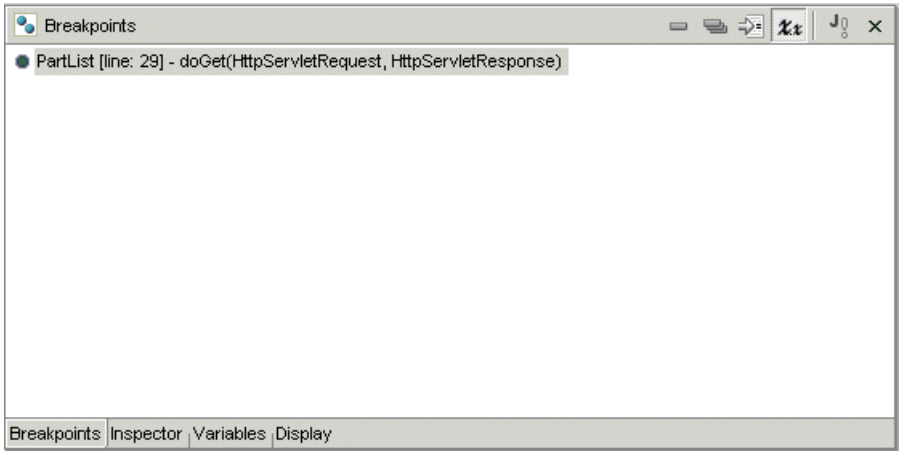


Figure 17-7 Debugging views

Next we will look at how to use the *Variables* view.

17.2.4 Watching variables

The Variables view displays the current values of the variables in the selected stack frame. Follow these steps to see how you can track the state of a variable.

Click the **Step Over** icon to execute the current statement. Note that a new variable `partListResult` has been added to the Variables view. Expand it until you can see the `elementCount` variable. It will currently have the value 0.

Click **Step Over** again. The `elementCount` variable is now 3, showing that 3 matching parts were returned from the database query (Figure 17-8).

If you want to test the code with some other value for this variable, you can change it by selecting **Change Variable Value** from its context menu.

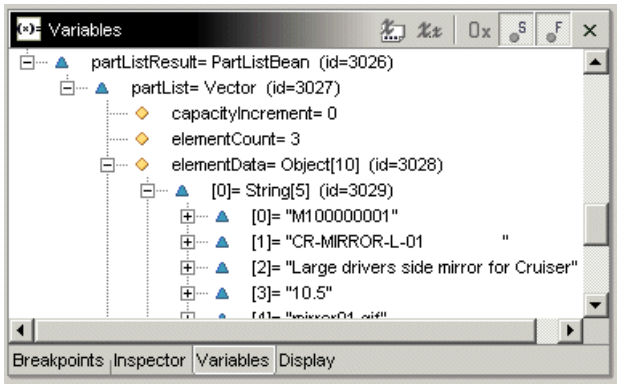


Figure 17-8 Displaying variables

4. If the program has not executed fully when you are done debugging, click **Resume** to run the process to the end or **Terminate** to end it.

17.3 Manually debugging on a remote WebSphere AEs

It is possible to connect to and debug a Java program that has been launched in debug mode on a remote application server, and the application server has been configured to accept remote connections. Debugging a remote program is similar to debugging a local Java program, except that the program has already been launched and could be running on a remote host.

17.3.1 Starting the remote server

You need to start WebSphere AEs in debug mode. The following command can be used:

```
./startServer.bat -debugEnabled -jdwpPort 3001
```

17.3.2 Starting Application Developer in remote mode

From the drop-down menu on the **Debug** button in the workbench toolbar, select **Debug—>Remote Application** (Figure 17-9).

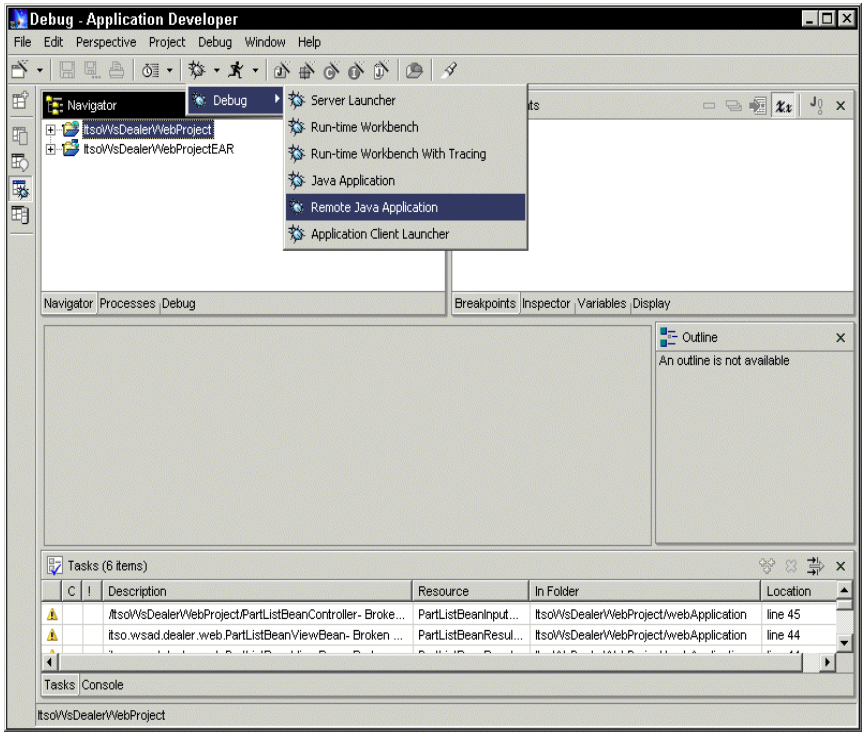


Figure 17-9 Starting the remote application

The Remote Java Application dialog will be displayed (Figure 17-10).

- In the **Host** field, type the IP address or name of the host where the Java program is running.

  If the program is running on the same machine as Application Developer, type `localhost`.

- In the **Port** field, type the port on which the remote Java Virtual Machine (VM) is accepting connections. Generally, this port is specified when the remote VM is launched.

- The **Allow termination of remote VM** flag is a toggle that determines whether the Terminate command is enabled in the debugger. Select this option if you want to be able to terminate the VM to which you are connecting.

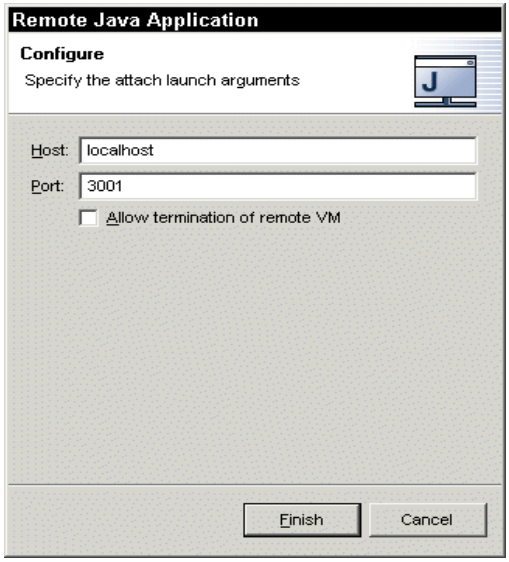


Figure 17-10 Remote server configuration

Click **Finish** when you have entered the required parameters.

The launcher attempts to connect to a VM at the specified address and port, and the result is displayed in the Debug view. If the launcher is unable to connect to a VM at the specified address, an error message appears.

17.3.3 Disconnecting from the remote VM

To disconnect from a VM that was connected to with the Remote Java Application launcher, select the VM in the Debug view and click the **Disconnect** button in the view's toolbar (Figure 17-11).

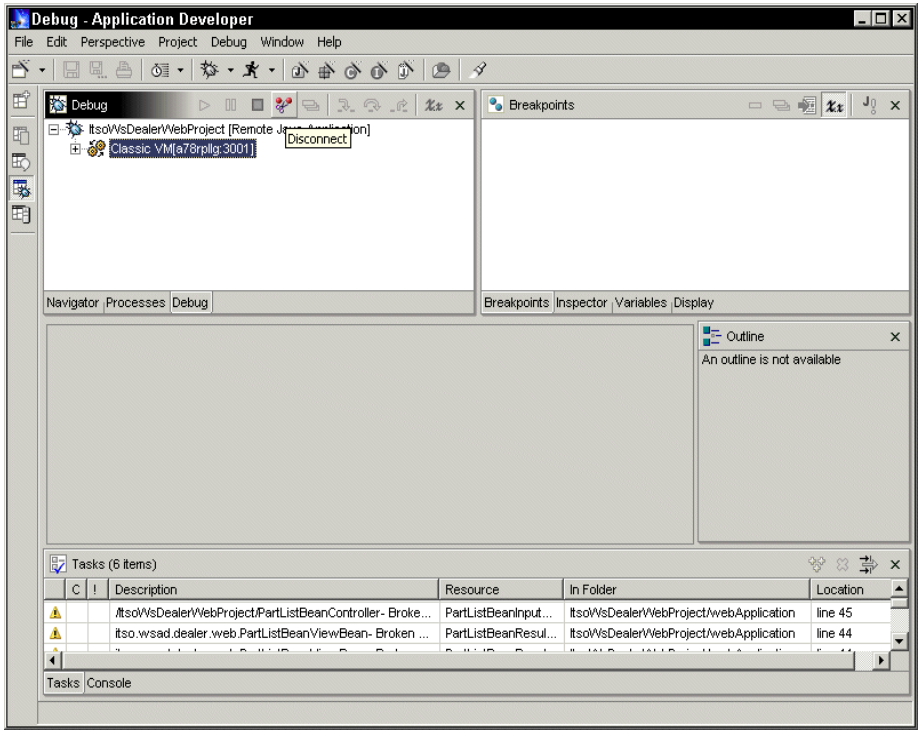


Figure 17-11 Disconnect from remote VM

Communication with the VM is terminated, and all threads in the remote VM are resumed. Although the remote VM continues to execute, the debug session is now terminated.

18

JUnit test framework

JUnit is an Open Source testing framework for Java and is included in Application Developer package. It provides a simple way of expressing how you intend your code to work. In this chapter, we describe how to setup JUnit for use in the workbench and how to write tests with JUnit.

© Copyright IBM Corp. 2002

18.1 What is JUnit

JUnit is an open source testing framework that is used to develop and execute unit tests in Java. It was written by Erich Gamma, one of four notable authors, who wrote the classic book *Design Patterns*; and Kent Beck, who has also written extensively about object development and first described the eXtreme Programming (XP) software development process.

A good starting point for finding information about JUnit on the Web is the JUnit Web site:

```
http://www.junit.org/
```

This site contains documentation and links, as well as a free download that includes both the JUnit source and compiled code.

18.1.1 Unit testing

Unit tests are informal tests that are generally executed by the developers of the application code. They are often quite low-level in nature, and test the behavior of individual software components such as individual Java classes, servlets or EJBs.

Because unit tests are usually written and performed by the application developer, they tend to be “white-box” in nature, that is to say they are written using knowledge about the implementation details and test specific code paths. This is not to say all unit tests have to be written this way, one common practice is to write the unit tests for a component based on the component specification *before* developing the component itself. Both approaches are valid and you may want to make use of both when defining your own unit testing policy.

18.1.2 Why unit testing?

On the face of it this is a question with a straightforward answer. We test to find defects in our code, and to verify that changes we have made to existing code do not break that code.

Perhaps it is more useful to look at the question from the opposite perspective, that is to say, why do developers *not* perform unit tests? In general the simple answer is because it is too hard, and because nobody forces them to. Writing an effective set of unit tests for a component is not a trivial undertaking. Given the pressure to deliver that many developers find themselves subjected to, the temptation to postpone the creation and execution of unit tests in favour of delivering code fixes or new functionality is often overwhelming.

In practice, this usually turns out to be a false economy—developers very rarely deliver bug-free code, and the discovery of code defects and the costs associated with fixing them are simply pushed further out into the development cycle. This is inefficient—the best time to fix a code defect is immediately after the code has been written, while it is still fresh in the developer's mind. Furthermore, a defect discovered during a formal testing cycle must be written up, prioritized and tracked—all of these activities incur cost, and may mean that a fix is deferred indefinitely, or at least until it becomes critical.

Based on our experience, we believe that encouraging and supporting the development and regular execution of unit test cases ultimately leads to significant improvements in productivity and overall code quality. The creation of unit test cases do not need to be a burden—developers often find the intellectual challenge quite stimulating and ultimately satisfying. The thought process involved in creating a test can also highlight shortcomings in a design which may not otherwise have been identified in a situation where the main focus is on implementation.

We recommend that you take the time to define a unit testing strategy for your own development projects. A simple set of guidelines and a framework that makes it easy to develop and execute tests will pay for itself surprisingly quickly.

18.1.3 Benefits of a unit testing framework

Once you have decided to implement a unit testing strategy in your project, the first hurdles to overcome are the factors that dissuade developers from creating and running unit tests in the first place. A testing framework can help by:

- Making it easier to write tests
- Making it easier to run tests
- Making it easier to rerun a test after a change

Tests are easier to write, because a lot of the infrastructure code that you require to support every test is already available. A testing framework also provides a facility that makes it easier to run and re-run tests, perhaps via a GUI. The more often a developer runs tests, the quicker problems can be located and fixed, because the delta between the code that last passed a unit test and the code that fails the test is smaller.

Testing frameworks also provide other benefits:

- **Consistency**: Because every developer is using the same framework, all of your unit tests will work in the same way, can be managed in the same way, and report results in the same format.

- **Maintenance**: Because a framework has already been developed and is probably already in use in a number of projects you spend less time maintaining your testing code.
- **Ramp-up time**: If you select a popular testing framework, you may find that new developers coming into your team are already familiar with the tools and concepts involved.
- **Automation**: A framework may offer the ability to run tests unattended, perhaps as part of a daily or nightly build.

Automatic builds: A common practice in many development environments is the use of daily builds. These automatic builds are usually initiated in the early hours of the morning by a scheduling tool such as `cron`, which is standard on UNIX systems. Similar tools are available for Windows environments.

18.1.4 How to test

A unit test is a collection of tests designed to verify the behavior of a single unit within a class. JUnit tests your class by scenario, and you will need to create a testing scenario that uses the following elements:

- Instantiate an object
- Invoke methods
- Verify assertions

This simple test case tests the result count of database query.

```
//Tester method
public void testPartListBean(){
       //instantiate
       PartListBean plb = new PartListBean();
       //invoke two methods
       plb.select(“IRRO”);
       int count = plb.getpartList().elementCount();
       //verify an assertion
       assertEquals(0,count);
}
```

In JUnit each test is implemented as a method that should be declared as *public void* and take no parameters. This method is then invoked from a test runner defined in a different package. If the test method name begins with *test...*, the test runner will find it automatically and run it. This way, if you have a large number of test cases, there is no need to explicitly define all the test methods to the test runner.

18.1.5 TestCase class

The core class in the JUnit test framework is *JUnit.framework.TestCase.* All of our test cases inherit from this class.

```
import junit.framework.*;
public class PartListBeanTester extends TestCase {
   /**
    * Constructor for PartListBeanTester
    */
   public PartListBeanTester(String name) {
      super(name);
   }
}
```

All test cases must have a constructor with a string parameter. This is used as a test case name to display in the log.

To run this test, use TestRunner as follows.

```
public static void main (String[] args) {
   junit.textui.TestRunner.run (PartListBeanTester);
}
```

18.1.6 TestSuite class

Test cases can be organized into test suites, managed by the `junit.framework.TestSuite` class. JUnit provides tools that allow every test in a suite to be run in turn and to report on the results.

To add new test case you use addTest method of TestSuite.

```
TestSuite suite = new TestSuite();
suite.addTest(new PartListBeanTester("testPartListBeanConnection"));
suite.addTest(new PartListBeanTester("testPartListBean"));
```

Alternatively, a TestSuite can extract the tests to be run automatically. To do so you pass the class of your TestCase class to the TestSuite constructor.

```
TestSuite suite = new TestSuite(PartListBeanTester.class);
```

This constructor creates a suite containing all methods starting with "test" and that take no arguments.

18.2 Installing JUnit

To use JUnit in the Application Developer Workbench, you need to import the JUnit package.

The jar file is:

`/plugins/org.eclipse.jdt.ui.examples.projects/archive/junit/folder/`**`junit37src.jar.`**

Create a Java project, name it `JUnit`, then import the jar file into the new project (Figure 18-1).

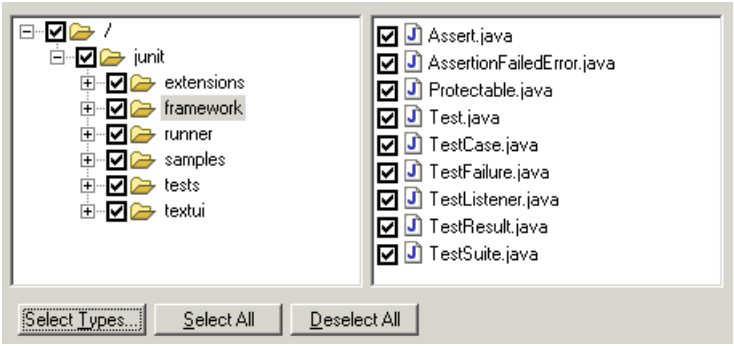


Figure 18-1 Importing JUnit

You are ready to start using the JUnit framework

18.3 Creating the test case

We will show how to create a test case to test our PartListBean class. PartListBean class is the business method of the PartList Web application and is described in Chapter 8, “Creating Web applications with dynamic content” on page 203.

PartListBean is invoked by the PartList servlet and PartList.jsp. The servlet invokes the bean and it connects to the database, searches it, and creates the result set. This is the code of the method that invokes the bean:

```
public void doGet(
   javax.servlet.http.HttpServletRequest request,
   javax.servlet.http.HttpServletResponse response)
   throws javax.servlet.ServletException, java.io.IOException {
   try {
      String resultPage = "/PartList.jsp";
      String partialName = request.getParameter("partialName");

      PartListBean partListResult = new PartListBean();
      partListResult.select(partialName);
      request.setAttribute("partListResult", partListResult);
      //Forward the request to the next page
```

```
            RequestDispatcher dispatch =
                      request.getRequestDispatcher(resultPage);
            dispatch.forward(request, response);
        } catch (Throwable e) {...}
    }
```

The JSP is then dispatched, and it invokes the bean again to retrieve the search results:

```
<jsp:useBean id="partListResult" type="itso.wsad.dealer.web.PartListBean"
scope="request"/>
<%! int i; String[] row; %>
<TABLE border="1">
    <TBODY>
        <TR>
            <TH>Number</TH>
            <TH>Name</TH>
            <TH>Description</TH>
            <TH>Weight</TH>
            <TH>Image</TH>
        </TR>
     <% try {
        for (i=0; ; i++) {
          row = (String[])partListResult.getPartList().elementAt(i); %>
        <TR>
            <TD> <%= row[0] %> </TD>
            <TD> <%= row[1] %> </TD>
            <TD> <%= row[2] %> </TD>
            <TD> <%= row[3] %> </TD>
            <TD> <img src="images/<%= row[4] %>"> </TD>
        </TR>
     <%    }
        } catch (Exception e) {} %>
    </TBODY>
</TABLE>
```

To test the bean we will run it several times and check that the result count is correct.

18.3.1 Test case class definition

We will call our test class PartListBeanTester and it must inherit from *junit.framework.TestCase*. We created a separate project for the test case, because we don't want to include these test modules into the build package. The outline of the class, including the required constructors, is as follows.

```
import junit.framework.*;

public class PartListBeanTester extends TestCase {
```

```
    public PartLstBeanTester(String name){
        super(name);
    }
}
```

18.3.2 The setUp and tearDown methods

We will run several tests in one test case. To make sure there are no side effects between test runs, the JUnit framework provides the *setUp* and *tearDown* methods. Every time the test case is run, setUp will be called at the start and tearDown at the end of the run. We have two global variables that should be cleaned up before starting the test:

```
//Holding a latest exception when it happened
private Exception ex;
//Test target
private PartListBean plb;
```

In the setUp method, we clean up the exception log, so that it does not affect another test.

```
public void setUp(){
      ex = null;
}
```

In the tearDown method, we attempt to disconnect from the database. PartListBean *should* have disconnected from the database when it has completed its work, but we are guarding against the possibility that it ended before disconnecting.

```
public void tearDown(){
    if (plb!= null){
        try{
            plb.disconnect();
        }catch(Exception e){
            //do nothing.
        }
    }
}
```

18.3.3 Testing body

We now create a common method to invoke PartListBean and get the results. This method can be used by several different test case methods. The common method invokes PartListBean with a search criteria and returns a count or matches found on the database. If an exception occurs, we save it in the global variable *ex*.

```
        plb = new PartListBean();
```

```
        try{
            plb.select(param);
            count = plb.getPartList().size();
        }catch(Exception e){
            ex = e;
            count = -1;
        }
        return count;
    }
```

18.3.4 Creating test methods

Two test methods were created. One uses “IRRO” as the search argument. To pass the test it should return three matches. If it fails with an exception it is reported as a failure. The second method uses “GHT” as an argument. This search should return only one match.

```
    public void testGetPartsCountByIRRO(){
        ex = null;
        int count = invokePartListBean("IRRO");
        if(ex != null) {
            fail(ex.toString());
        }else{
            assertEquals(3,count);
        }
    }

    public void testGetPartsCountByGHT(){
        ex = null;
        int count = invokePartListBean("GHT");
        if(ex != null) {
            fail(ex.toString());
        }else {
            assertEquals(“Count does not match. It should be zero.”0,count);
        }
    }
```

The assertEquals and fail methods are provided by the JUnit framework. JUnit provides a number of methods that can be used to assert conditions and fail a test if the condition is not met. These methods are inherited from the class *junit.framework.Assert* (see Table 18-1). All of these methods include an optional `String` parameter that allows the writer of a test to provide a brief explanation of why the test failed—this message is reported along with the failure when the test is executed (ex. assertEquals(String message, object expected, object actual);).

Table 18-1 JUnit assert methods

Method name	Description
`assertEquals`	Assert that two objects or primitives are equal. Compares objects using `equals`, and compares primitives using `==`.
`assertNotNull`	Assert that an object is not null
`assertNull`	Assert that an object is null
`assertSame`	Assert that two objects refer to the same object. Compares using `==`.
`assertTrue`	Assert that a `boolean` condition is `true`
`fail`	Fails the test

18.3.5 Create a TestSuite

A TestSuite was used to run our test cases. `PartListBeanTester.class` is passed as a parameter when creating the TestSuite, and it will invoke both our test methods:

```
public static Test suite() {
   return new TestSuite(PartListBeanTester.class);
}
```

18.3.6 Implementing TestRunner

The JUnit framework provides both text and GUI test runner tools, which can run your tests and report the results. However, by importing *junit37src.jar*, we only have access to the text version. The text-based test runner included with JUnit is started from the `junit.textui.TestRunner` class. To make the testcase an executable, we added a main method. This executes the run method of the TestRunner class with the TestSuite we created earlier as the argument.

```
public static void main (String[] args) {
   junit.textui.TestRunner.run (suite());
}
```

18.4 Running the test case

Before we can run the test tool, we must first make sure that all the classes we need are in the class path. In this case we need to include the JUnit code and ItsoWsDealerWebProject.

Figure 18-2 shows the output from a test run. We can see that there was one success and one failure. The failure occurred when running testGetPartsCountByGHT. The expected result was 1 but 0 was returned instead. Each dot (.) in the output represents the start of a test. We have two tests in our test case, so there are two dots. An “F” indicates a failure so we got one fail. Once all the tests have completed the test runner shows how long they took to run and a summary of the results.

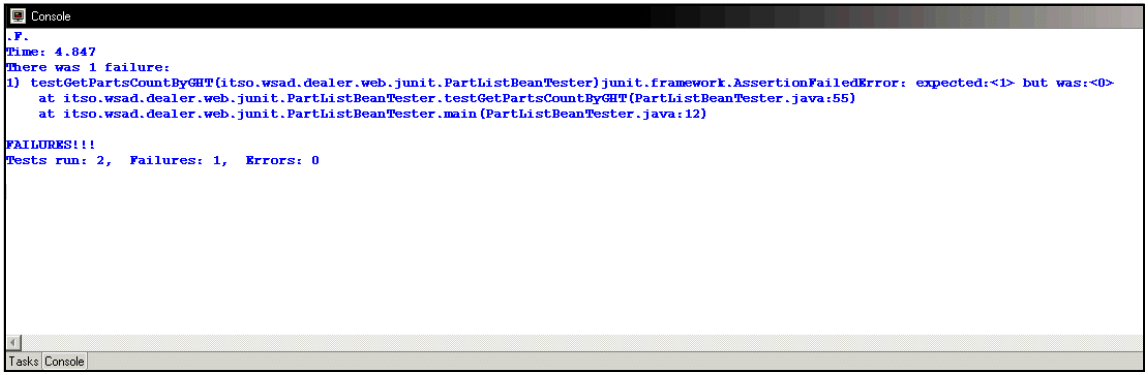


Figure 18-2 Failure when running test

A test is considered to be *successful* if the test method returns normally. A test *fails* if one of the methods from the `Assert` class signals a failure. An *error* indicates that an unexpected exception was raised by the test method, or the `setUp` or `tearDown` method invoked before or after it.

Once we have corrected the error, the output will look as in Figure 18-3.

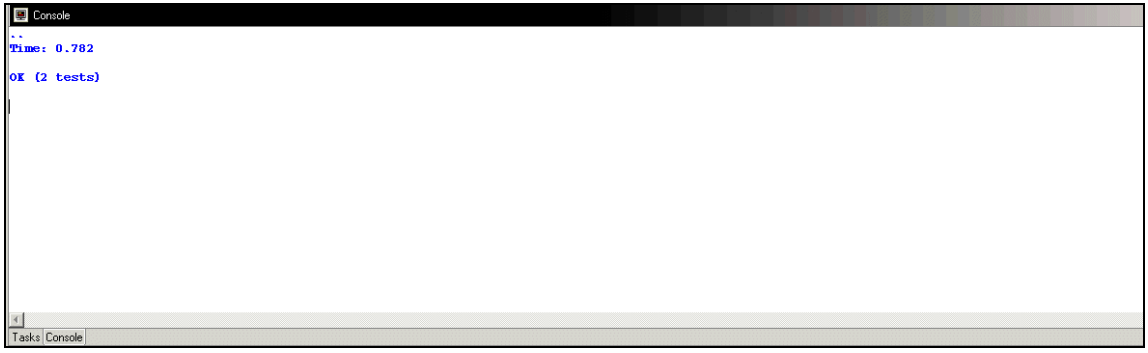


Figure 18-3 Test completed without errors

18.4.1 Testing the servlet

We tested our servlet and JSP using the following TestCase. Two different parameters are used and the search is repeated ten times. The test case submits the parameter using the POST method and gets a result from the JSP.

```
package itso.wsad.sg246585.junit;

import java.io.*;
import java.net.*;
import junit.framework.TestCase;
import junit.textui.TestRunner;

public class PartListTester extends TestCase {

   public PartListTester(String name) {
      super(name);
   }

   public static void main(String[] args) {
      junit.textui.TestRunner.run (PartListTester.class);
   }

   public void testPartListServlet(){
      int cnt = 0;
      for(int i=0; i<10; i++){
         cnt=commonTestUnit("http://localhost:8080/ItsoWsDealerWeb/PartList",
                                                    "partialName=IRRO");
         assertEquals(1456, cnt); //check the result count

         cnt=commonTestUnit("http://localhost:8080/ItsoWsDealerWeb/PartList",
                                                    "partialName=AKE");
         assertEquals(904, cnt);
      }
   }

   private int commonTestUnit(String url,String postData){
      int count=0;
      try{
         URL u = new URL(url);
         URLConnection uc = u.openConnection();
         uc.setDoOutput(true);
         DataOutputStream dos = new DataOutputStream(uc.getOutputStream());
         dos.writeBytes(postData);
         dos.close();
         count = uc.getContentLength();
      }catch(Exception e){
         fail(e.toString());
      }
```

```
        return count;
    }
}
```

19

Deploying your Web application

Deployment of a Web application can be done manually or automatically using Application Developer.

This chapter describes the following:

- Manual deployment
  - Exporting your EAR from Application Developer
  - Installing the EAR file on WebSphere AEs
  - Starting the WebSphere AEs admin GUI
  - Installing the EAR on WebSphere AEs
  - Configuring your server
  - Testing the application
- Automatic deployment to a remote server
  - Creating a remote server instance
  - Publishing to remote server
  - Testing the application

© Copyright IBM Corp. 2002

19.1 Manual deployment

This section explains how to deploy the application manually to WebSphere AEs.

19.1.1 Exporting your project from Application Developer

Complete the following steps to export an EAR file.

1. From the **File** menu, select **Export**.
2. Select the **EAR file export** wizard and click **Next**.

   The window as shown in Figure 19-1 will appear.

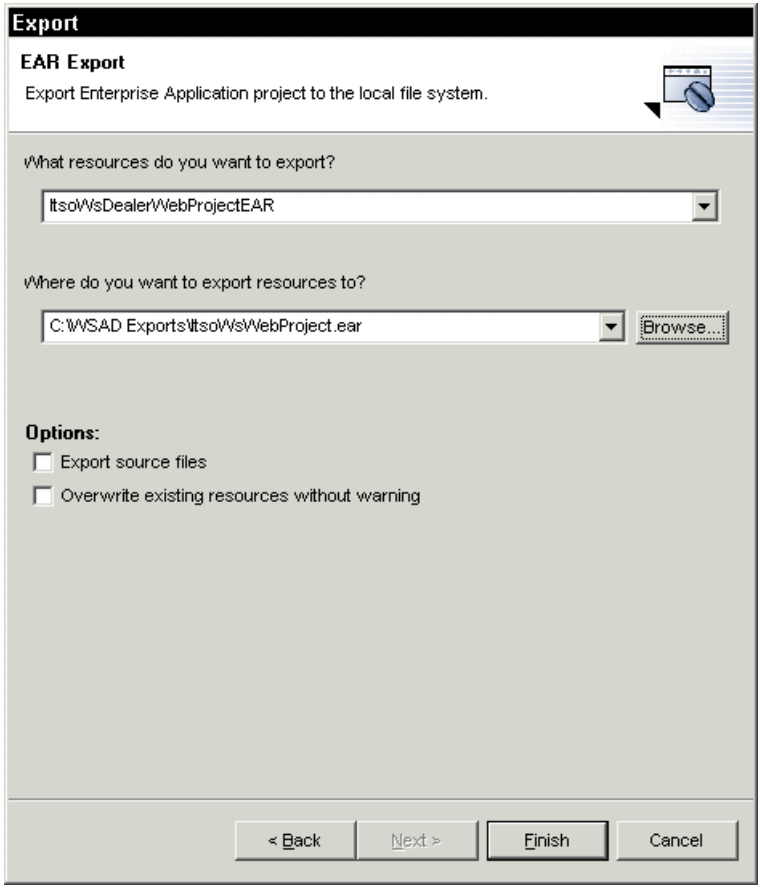


Figure 19-1 Exporting a project EAR file

In this window you can select the EAR project that you want to export as well as select the location where you want to export.

3. You can further specify additional options as appropriate using the **Options** check boxes. To export the source files, select the **Export source files** check box. If you are exporting to an existing file and you do not want to be warned about overwriting it, select the **Overwrite existing files without warning** checkbox.
4. Click **Finish**.

19.1.2 Installing the EAR file on WebSphere AEs

This topic discusses how to install the EAR file on WebSphere.

After exporting the EAR file, ftp or copy the EAR file to the *../WebSphere/AppServer/installableApps* directory in the WebSphere AEs product directory structure.

The EAR file can be installed by either using the *WebSphere Admin Console* or manually by the command line option *SEAppInstall*. An example of manual installation for the ItsoWsWebProject would be:

```
SEAppInstall -install ../installableApps/ItsoWsWebProject.ear
```

19.1.3 Starting the WebSphere AEs Admin Console

Start the Application Admin Server by selecting **Start—>Programs—>IBM WebSphere Application Server V4.0 AES—>Start Admin Server**, on a Windows machine or use the startServer.sh script on a UNIX machine.

The server is started when the following message is shown in the console:

```
SPLO057I: The server Default Server is open for e-business.
Please review the server log files for additional information.
Standard output: C:\WebSphere\AppServer/logs/default_server_stdout.log
Standard error: C:\WebSphere\AppServer/logs/default_server_stderr.log
```

Once you have started the Application Server, start the Administrator's console by one of the two methods. Select **Start—>Programs—>IBM WebSphere Application Server V4.0 AES—>Administrator's Console**, on a Windows NT machine, or by accessing it from a browser by typing `http://localhost:9090/admin/`. Enter a user ID in the window and press the submit button.

Note: This user ID does not need to be a valid User ID on the system. It is only used only for tracking user-specific changes to the configuration data.

19.1.4 Installing the EAR

This topic discusses how to install an EAR file in WebSphere AEs using the Admin Console. During this task, you will install the application files (.ear, .jar, and .war).

To install an application:

1. Expand the tree on the left side of the console to locate **Nodes —> hostname —> Enterprise Applications**.
2. Select the **Enterprise Applications**.

   The right side of the console should display the list of zero or more installed applications (as .ear files).
3. Click the **Install** button displayed on the right side of the console, above the list of installed applications.

   The screen in Figure 19-2 will be displayed.

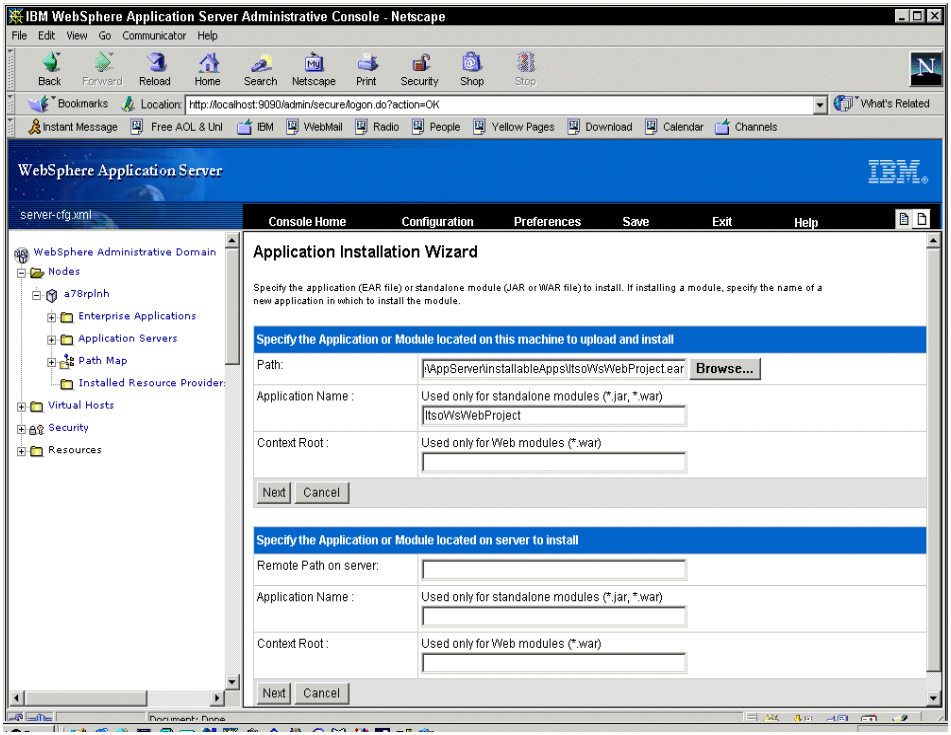

IBM WebSphere Application Server Administrative Console - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:9090/admin/secure/logon.do?action=OK
What's Related
Instant Message Free AOL & Unl IBM WebMail Radio People Yellow Pages Download Calendar Channels
WebSphere Application Server
IBM
server-cfg.xml
Console Home Configuration Preferences Save Exit Help
WebSphere Administrative Domain
Nodes
a78rplnh
Enterprise Applications
Application Servers
Path Map
Installed Resource Provider
Virtual Hosts
Security
Resources
Application Installation Wizard
Specify the application (EAR file) or standalone module (JAR or WAR file) to install. If installing a module, specify the name of a new application in which to install the module.
Specify the Application or Module located on this machine to upload and install
Path:
\AppServer\installableApps\ItsoWsWebProject.ear
Browse...
Application Name :
Used only for standalone modules (*.jar, *.war)
ItsoWsWebProject
Context Root :
Used only for Web modules (*.war)
Next Cancel
Specify the Application or Module located on server to install
Remote Path on server:
Application Name :
Used only for standalone modules (*.jar, *.war)
Context Root :
Used only for Web modules (*.war)
Next Cancel

Figure 19-2 Application installation wizard

Here you need to specify the location of the application or module, referring to the application property reference as needed to fill in the field values, and the application name.

Note: Use the first set of fields if the console and application files are on the same machine (whether or not the server is on that machine, too). Use the second set of fields (including Remote Path...) if the application files already reside on the same machine as the server, and the console is being run from a remote machine. See the field help for further discussion of the remote file path.

4. Click **Next**.
5. On the next page select the **`Virtual Host`** Name.
6. Follow the instructions on the resulting task wizard. Depending on the components in your application, various panels will be displayed:

   Modify **Role to User Mapping** (Valid for all applications).

   – Modify **EJB Run as Role to User Mapping** (Valid for applications containing EJB modules with one or more entity beans that use the IBM deployment descriptor extension for Run As Settings, Run As Mode, Run As Specified Identity).

   – Modify **EJB to JNDI Name Mapping** (Valid for applications containing EJB modules).

   – Modify **EJB Reference to JNDI Name Mapping** (Valid for applications containing EJB modules with EJB references in their deployment descriptors).

   – Modify **Resource Reference to JNDI Name Mapping** (Valid for applications containing Web modules with Resource references in their deployment descriptors).

   – Specify **Virtual Host Mapping** (Valid for applications containing Web modules) -- includes JSP precompilation option.

   – Specify **CMP Data Source Binding** (Valid for applications containing EJB modules with container managed entity beans).
7. Click **Finish** when you have completed the wizard.

Note: Be patient if the application you are installing contains EJB modules for which code must be generated for deployment. This step can take a while.

8. Verify that the new application is displayed in the tree on the left side of the console. It should be located at **Nodes—>hostname—>Enterprise Applications** (Figure 19-3).

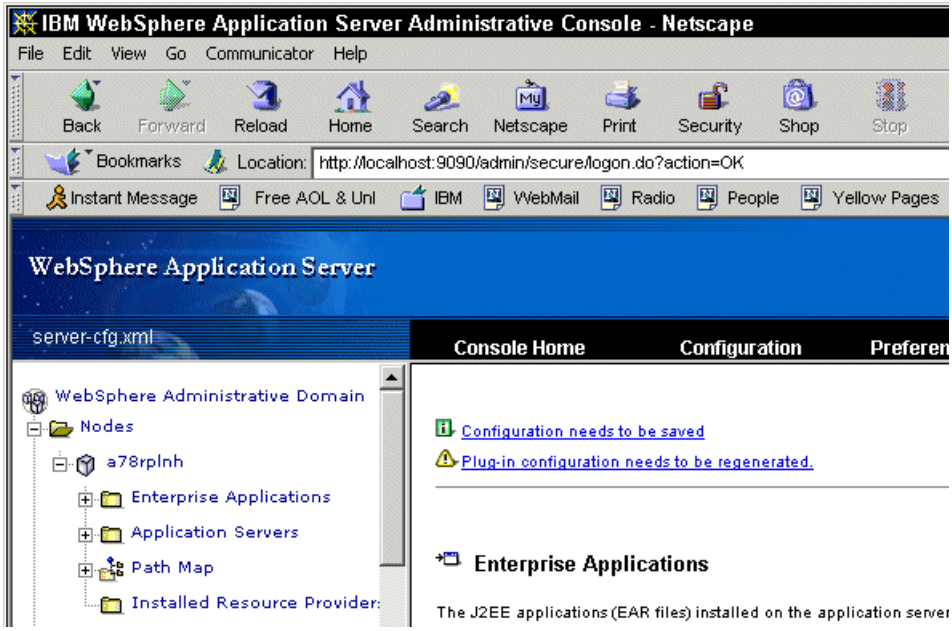

IBM WebSphere Application Server Administrative Console - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:9090/admin/secure/logon.do?action=OK
Instant Message Free AOL & Unl IBM WebMail Radio People Yellow Pages
WebSphere Application Server
server-cfg.xml
Console Home Configuration Preferen
WebSphere Administrative Domain
Nodes
a78rplnh
Enterprise Applications
Application Servers
Path Map
Installed Resource Provider
Configuration needs to be saved
Plug-in configuration needs to be regenerated.
Enterprise Applications
The J2EE applications (EAR files) installed on the application server

Figure 19-3 After application deployment

9. To save your configuration click the **Save** button (Figure 19-4).

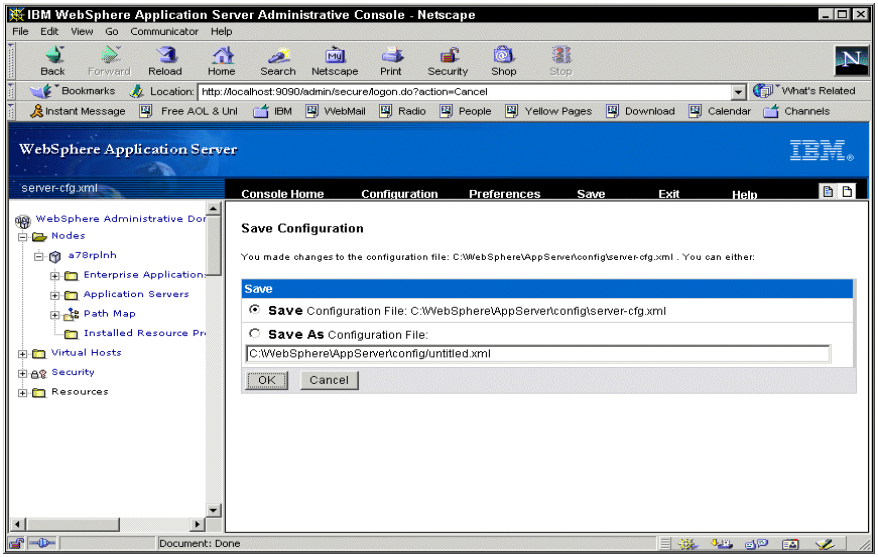


Figure 19-4 Saving configuration

10. Select the application and click the **Start** button.

    (**Optional**) To have the configuration take effect:

    – Stop the Application server.
    – Start the Application server again.
    – Stop the HTTP Web server.
    – Start the HTTP Web server again.

Note: If you installed an application while the server was running, the newly installed application and its modules can be viewed in the list of installed Enterprise Applications, from which applications and their modules can be started, stopped, and restarted. However, the application and modules will remain in a "cannot be run" state until the server is stopped and started again.

19.1.5 Testing the Application

You can now test your application on the remote WebSphere Application server.

To do so, start the Sample application start page (index) by typing `http://localhost:9080/ItsoWsDealerWebProject/` in your browser (Figure 19-5).

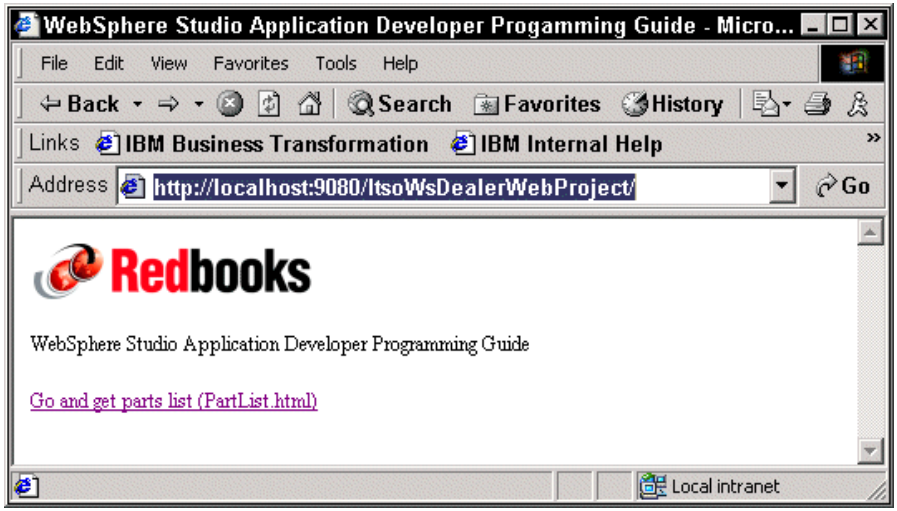

WebSphere Studio Application Developer Progamming Guide - Micro...
File Edit View Favorites Tools Help
Back Search Favorites History
Links IBM Business Transformation IBM Internal Help
Address http://localhost:9080/ItsoWsDealerWebProject/
Go
Redbooks
WebSphere Studio Application Developer Programming Guide
Go and get parts list (PartList.html)
Local intranet

Figure 19-5 Starting Parts application on WebSphere Application Server

19.2 Publishing to a remote server

These are some prerequisites to publishing to a remote server. With Application Developer, you can deploy the application to a remote server using a remote server instance and configuration. See Chapter 16, “Server instances and server configurations” on page 363 for more information. If you want to publish your projects remotely on WebSphere Application Server AEs, you must install the following applications on the remote server:

- IBM WebSphere Application Server Advanced Single Server Edition for Multiplatforms
- IBM Agent Controller
- (Optional) FTP server

Important: Ensure the IBM Agent Controller is running on the remote server and is inside a firewall.

Tip: In a team environment, care must be taken when publishing code to a remote server. Make sure the members of the team do not step on each other when deploying. It is a good idea to assign the deployment process to a lead developer to ensure correct code versions are deployed. This document does not define the details of the application build process.

19.2.1 Creating a remote server instance

From the Server view of the Server perspective, create the server configuration for the remote WebSphere Application Server AEs server. Select **New—>Server Instance** or **New—>Server Instance and Configuration**.

1. On the first page of the Create a New Server Instance wizard or the Create a New Server Instance and Server Configuration wizard (Figure 19-6), provide this information:
   - In the **Server Name** field, type a name for the new server.
   - In the **Folder** field, enter a folder name for the server.
2. Select **WebSphere Remote Server** as the instance type. The **Next** button is enabled allowing you to specify additional remote server information needed to transfer files remotely.
3. Click **Next**.

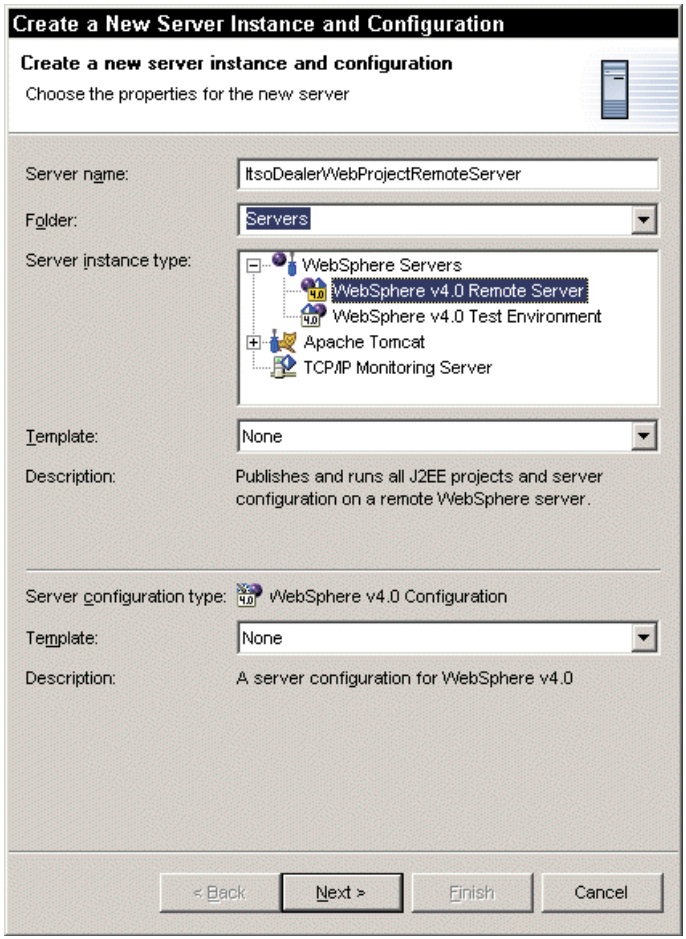


Figure 19-6 Creating a remote server instance

4. The second page of the server creation wizard opens allowing you to provide additional remote server instance information required for using the WebSphere Application Server remotely (Figure 19-7).

   Note: All paths on this page are seen from the remote machine.

   – In the **Host address** field, type the fully qualified DNS name or the IP address of the remote machine that WebSphere Application Server is running on. The field is pre-filled with the default address for the local host (127.0.0.1)

Tip: For more information about any of the fields on this and other wizards, select the field and then press **F1**.

– In the **WebSphere installation directory** field, type the path where you installed WebSphere Application Server on the remote machine. This path is the same as the WAS_ROOT path mappings as defined by the WebSphere server configuration.

If you have installed WebSphere Application Server in the default directory, use c:/WebSphere/AppServer directory as the WebSphere installation path.

If you select the **Use default WebSphere deployment directory** check box, then you want to use the default WebSphere deployment directory. The **WebSphere deployment directory** field is then pre-filled with the default value. Otherwise, in the **WebSphere deployment directory** field, type the path of the directory where the Web application and server configurations will be published. This directory is any existing directory that can be seen from the remote server machine.

If the directory E:/testdir resides on the remote machine, then type `E:/testdir` in this field.

If you are following WebSphere naming conventions and install WebSphere Application Server in the C:/WebSphere/AppServer directory, then the WebSphere deployment directory is C:/WebSphere/AppServer.

Note: When publishing to the remote server, the server configuration and the Web application will be published to a directory under the remote deployment directory called "config" and "installedApps" respectively.

If you select the **Use default WebSphere deployment directory** check box when creating a remote server instance and then publish using this instance, the default WebSphere Application Server server-cfg.xml file and plugin-cfg.xml files are replaced with the published version.

Optional: In the **DB2 driver location** field, type the DB2 location where the DB2 classes reside in the remote machine. If the default value is set in the Preference - WebSphere page, this field is pre-filled with the DB2 location.

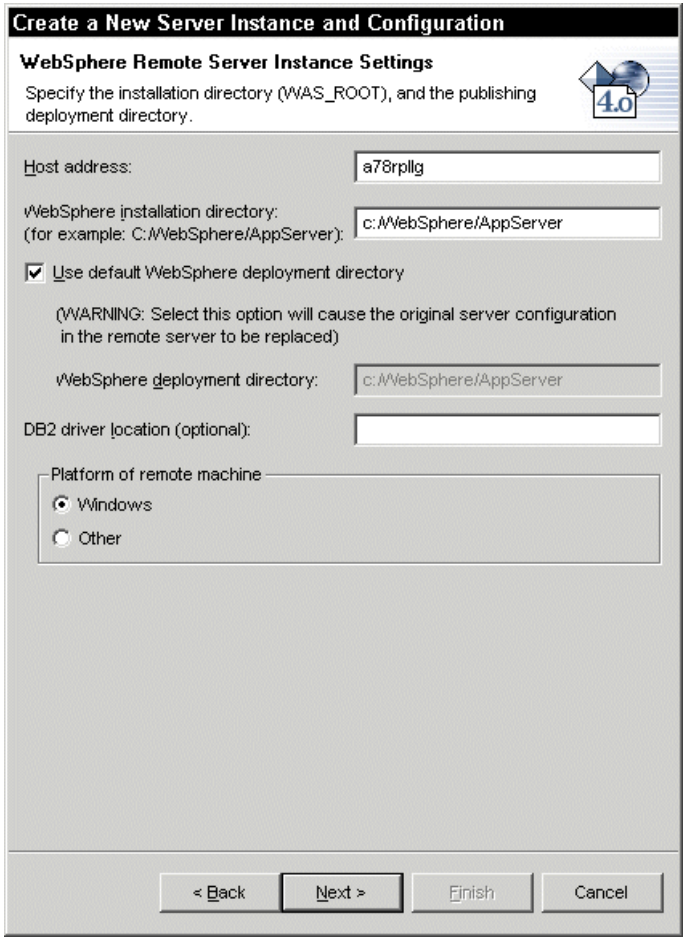


Figure 19-7 Remote server instance settings

5. Click **Next** again to display the third page of the server creation wizard.

   This page allows you define a remote file transfer instance. A *remote file transfer instance* contains information for transferring Web applications and server configurations to the remote server during publishing (Figure 19-8).

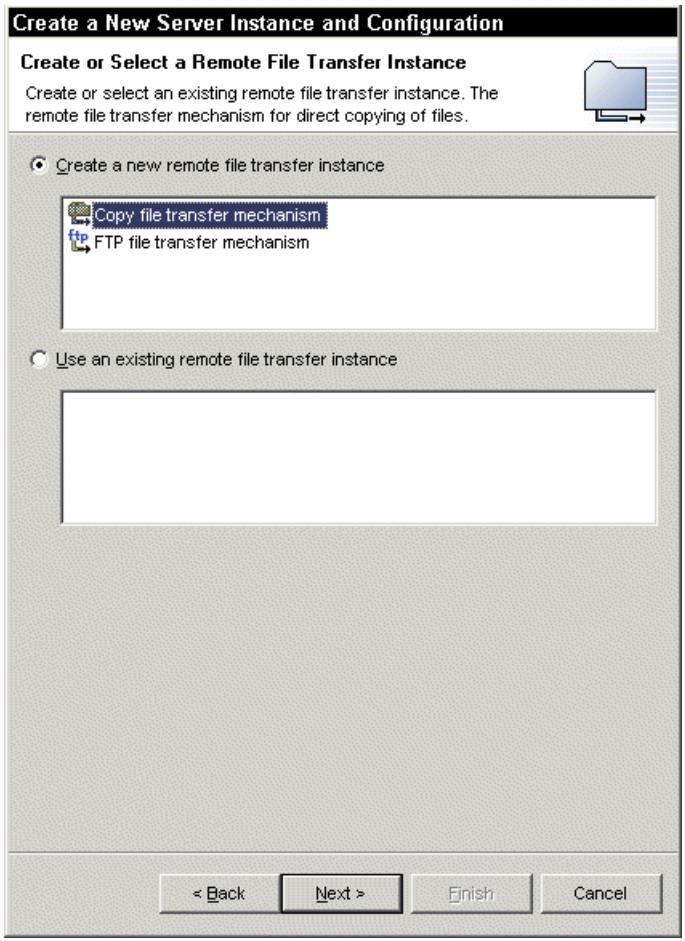

Create a New Server Instance and Configuration
Create or Select a Remote File Transfer Instance
Create or select an existing remote file transfer instance. The remote file transfer mechanism for direct copying of files.
Create a new remote file transfer instance
Copy file transfer mechanism
FTP file transfer mechanism
Use an existing remote file transfer instance
< Back
Next >
Finish
Cancel

Figure 19-8 Remote file transfer option

Select one of the following radio buttons:

- **Create a new remote file transfer instance**, defines a new set of parameters and environment settings needed to transfer files remotely.
- **Use an existing remote file transfer instance**, lists the already defined remote file transfer instances that you can use for transferring files remotely.

If **Create a new remote file transfer instance** radio button is selected:

- From the list, select one of the following:
  - **Copy file transfer mechanism**, to copy resources directly from one machine to another in the file system.

- **FTP file transfer mechanism**, to copy resources from one machine to another using File Transfer Protocol (FTP).

6. Click **Next** to display the fourth page of the server creation wizard.

   If **Use an existing remote file transfer instance** radio button is selected, the **Next** button is not enabled (Figure 19-9).

   – Select the remote file transfer instance that you want to use to transfer files remotely. Omit the next step.

   If **Copy file transfer mechanism** is selected, the next page of the wizard appears (Figure 19-9).

   – In the **Project folder** field, type the name of the project folder where the remote file transfer instance will reside.

   – In the **Remote file transfer name** field, type the name of the remote file transfer instance.

   Note: For more information about any of the fields on this and other wizards, select the field and then press **F1**.

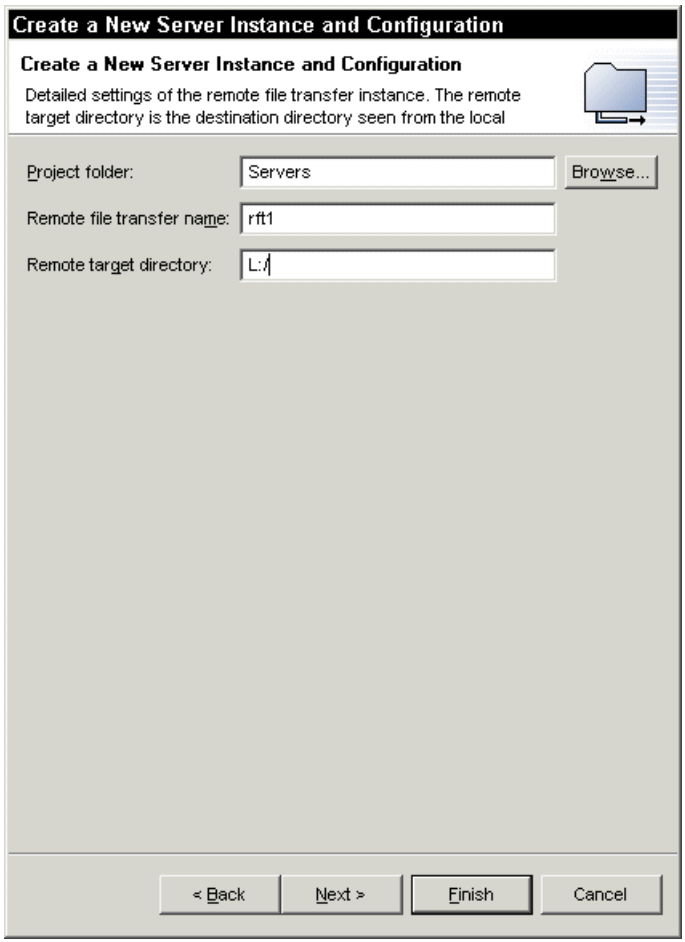

Create a New Server Instance and Configuration
Create a New Server Instance and Configuration
Detailed settings of the remote file transfer instance. The remote target directory is the destination directory seen from the local
Project folder:
Servers
Browse...
Remote file transfer name:
rft1
Remote target directory:
L:/
< Back
Next >
Finish
Cancel

Figure 19-9 Remote copy options

– In the **Remote target directory** field, type the remote target directory where you want your applications and server configuration published. This remote target directory is the one seen by the local machine. If WebSphere Application Server is installed on a different machine, then the remote target directory is the network drive that maps to the WebSphere deployment directory. If WebSphere Application Server is installed on the same machine as the workbench, then the remote target directory should be the same as the contents in the **WebSphere deployment directory** field.

If the remote Server is on the local machine, then `C:/WebSphere/AppServer` is the directory. If the remote Server is on a remote machine, then we use the mapped drive name for the shared directory (mapped as L:/ in our example).

The **Next** button is enabled only if you are creating a new server instance and server configuration together.

Click **Next** if you want to change the HTTP port number.

Click **Finish** to create a remote file transfer instance and a remote server instance. These instances will reside locally on your machine. The server instances appear in the Server Configuration view. The remote file transfer instances appears in the Navigator view.

7. If **FTP file transfer mechanism** is selected, the next page of the wizard appears Figure 19-10.
   - In the **Project folder** field, type the name of the project folder where the remote file transfer instance will reside.
   - In the **Remote file transfer name** field, type the name of the remote file transfer instance.
   - In the **Remote target directory** field, type the remote target directory where you want your application and server configuration published. This remote target directory points to the WebSphere deployment directory that is seen from the workbench using the FTP client program.

   If the WebSphere deployment directory is `C:/WebSphere/AppServer` and your FTP server route directory is `C:/`, then your remote target directory is /WebSphere/AppServer.

   Note: To determine whether a beginning slash is required, log on to the FTP server using a FTP client program, and then type the pwd command. If the results containing the default log on directory begins with a slash, then a slash is required prior to typing the remote target directory in the **Remote target directory** field.

   - In the **FTP URL** field, type the URL that is used to access the FTP server.
   - In the **User login** field and the **User password** field, type the FTP user ID and password that will be used to access the FTP server.
   - In the **Connection timeout** field, type the time (in milliseconds) that the workbench will wait this long while attempting to contact the FTP server before timing out.
   - Select the **Use PASV Mode (Passive Mode) to go through the firewall** check box, if you want to pass through a firewall provided that one is installed between your FTP server and the workbench.
   - Select the **Use Firewall** check box, if you want to use the firewall options.
   - To change the firewall options, select the **Use Firewall** check box, and then click **Firewall Settings**.

The **Next** button is enabled only if you are creating a new server instance and server configuration together. Click **Next** if you want to change the HTTP port number.

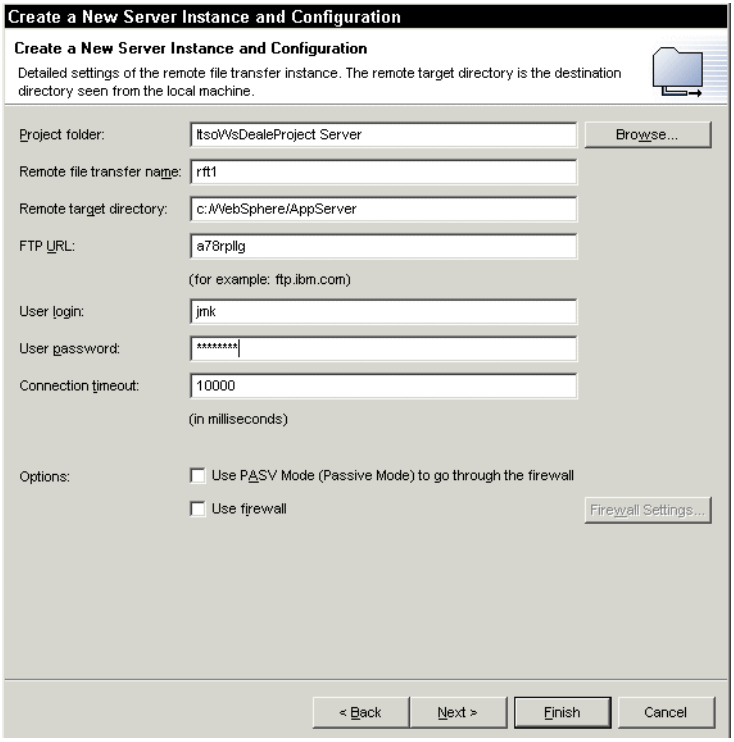


Figure 19-10 FTP configuration options

Click **Finish** to create a remote file transfer instance and a remote server instance. These instances will reside locally on your machine. The server instances appear in the Server Configuration view. The remote file transfer instances appears in the Navigator view.

8. Add the project to the new server configuration (Figure 19-11).

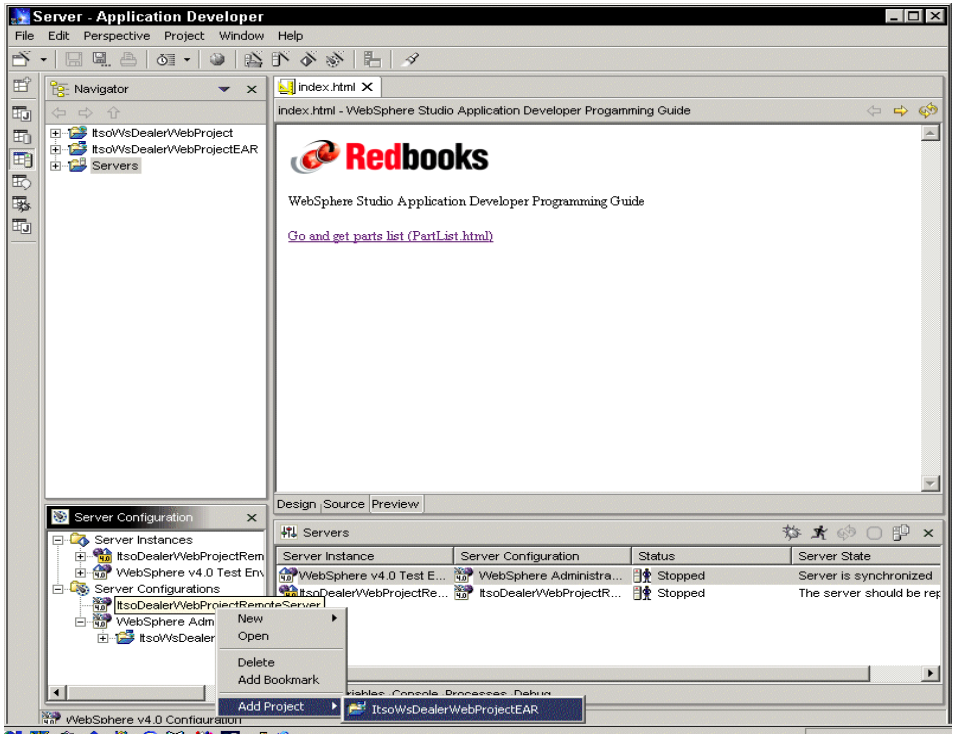


Figure 19-11 Add project to server configuration

9. Make sure the project is removed from the other server configurations (Figure 19-12).

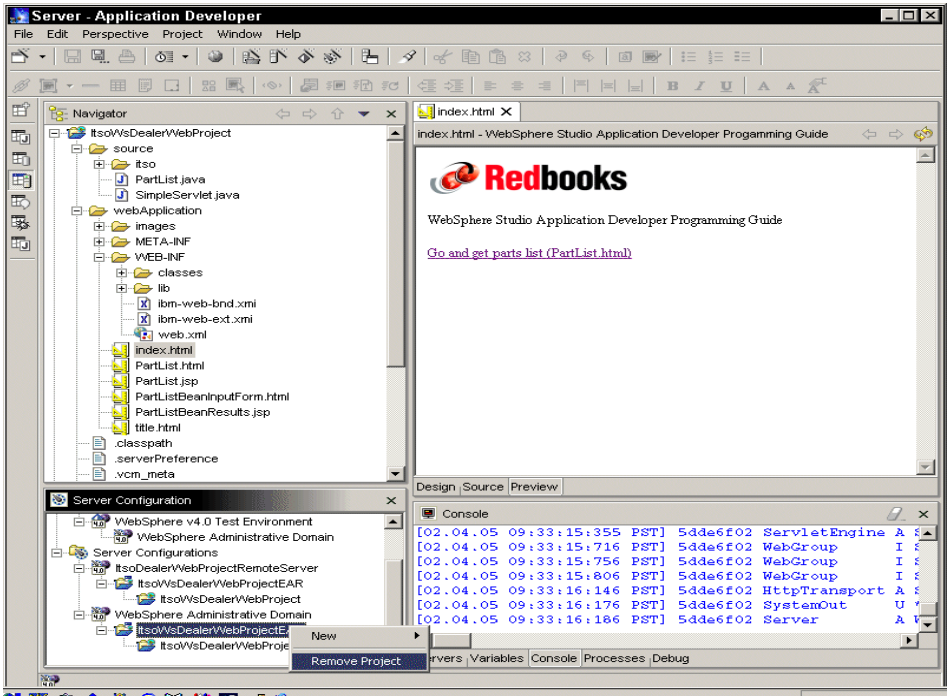


Figure 19-12 Remove project from other server configurations

19.2.2 Publishing to a remote server

Select **Run on Server** from the context menu of application entry view of the Server perspective.

On the remote machine you will see the new directory added to the installedApps directory (Figure 19-13).

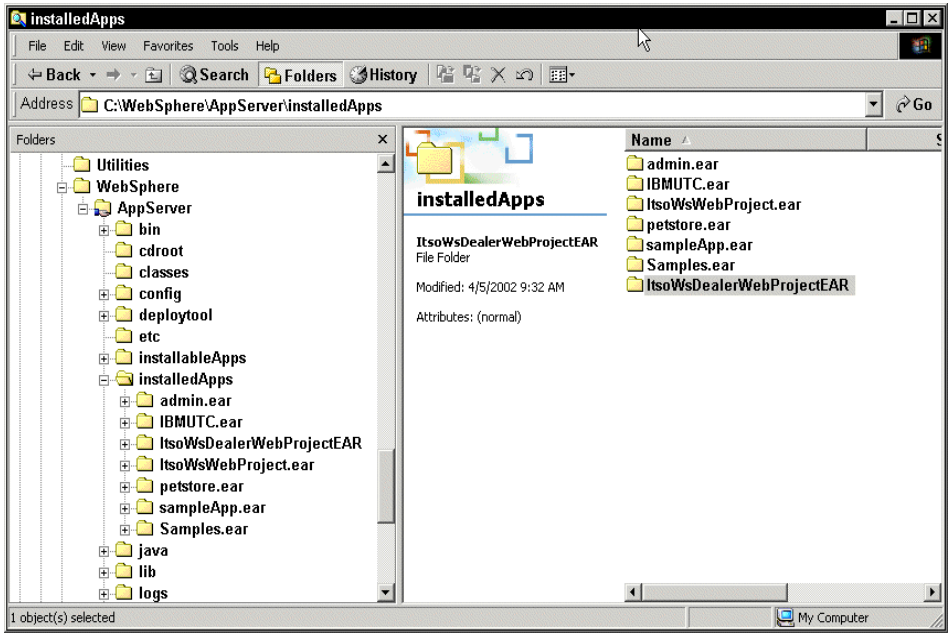


Figure 19-13 Directory listing on remote WebSphere AEs machine

19.2.3 Testing the application

You can now test your application on the remote Websphere Application server.

To do so, start the Sample application start page (index) by typing `http://localhost:9080/ItsoWsDealerWebProject/` in your browser (Figure 19-14).

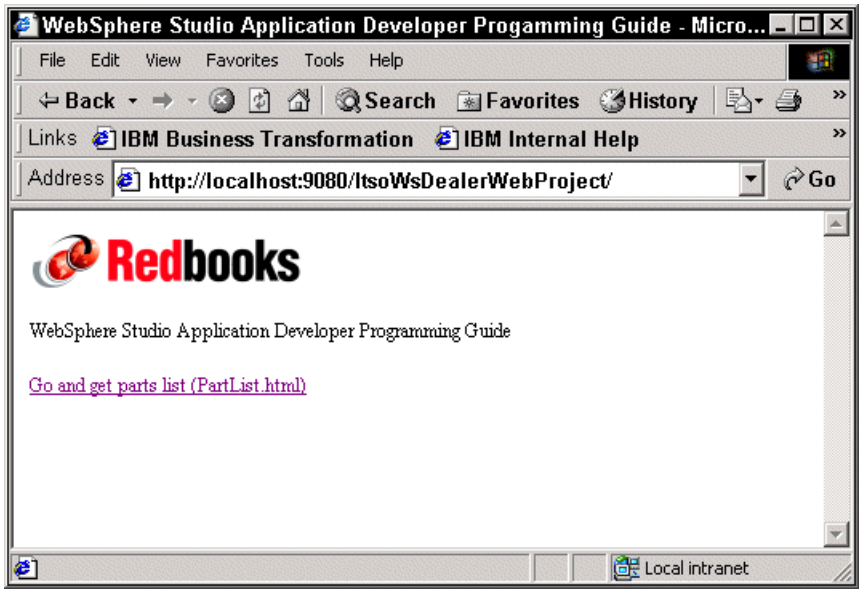

WebSphere Studio Application Developer Progamming Guide - Micro...
File Edit View Favorites Tools Help
Back Search Favorites History
Links IBM Business Transformation IBM Internal Help
Address http://localhost:9080/ItsoWsDealerWebProject/
Go
Redbooks
WebSphere Studio Application Developer Programming Guide
Go and get parts list (PartList.html)
Local intranet

Figure 19-14 Starting application on remote WebSphere Application Server

20

Building your application with Ant

Traditionally these tasks have been performed by shell scripts or batch files in UNIX or Windows environments, or by using tools such as *make*. While these approaches are still valid, developing Java applications—especially in a heterogeneous environment—introduces new challenges. A particular limitation is the close-coupling to a particular operating system inherent in using these tools.

© Copyright IBM Corp. 2002

20.1 What is Ant?

Ant is a Java-based, platform-independent, build tool that is an open-source, Jakarta project. Ant's function is similar to the *make* tool. As we described, it is java-based, and Ant does not utilize OS specific function, unlike the *make* tool. The build scripts are XML files containing targets and specifying the tasks. Ant comes with a large number of built-in tasks sufficient to perform many common build operations. You can learn about them in the Ant user manual. Of course Ant is well suited for building Java applications, but can be used for other build tasks as well. One of its important features is that you can use Java to write new Ant tasks to extend production build capabilities.

To find out more about the Jakarta project visit the Jakarta Web site at:

`http://jakarta.apache.org/`

The Ant Web site is located at:

`http://jakarta.apache.org/ant/`

This section provides a basic outline of the features and capabilities of Ant. For complete information you should consult the Ant documentation included in the Ant distribution or available on the Internet at:

`http://jakarta.apache.org/ant/manual/`

Note: Ant version 1.3 is shipped with Application Developer 4.0.2 or 4.0.3. You cannot update Ant under Version 4.0.3.

20.2 Ant build files

Ant uses XML *build files* to describe what tasks must be performed in order to build a project. The main components of a build file are:

- **project**: A build file contains build information for a single project. It may contain one or more *targets*.
- **target**: A target describes the *tasks* that must be performed to satisfy a goal, for example compiling source code into .class files may be one target, and packaging the .class files into a JAR file may be another target. Targets may depend upon other targets, for example the .class files must be up to date before you can create the JAR file. Ant will resolve these dependencies.
- **task**: A task is a single step that must be performed in order to satisfy a target. Tasks are implemented as Java classes that are invoked by Ant, passing parameters defined as attributes in the XML. Ant provides a set of

standard tasks, a set of optional tasks, and an API which allows you to write your own tasks.

- **property**: A property has a name and a value. Properties are essentially variables that can be passed to tasks via task attributes. Property values can be set inside a build file, or obtained externally from a property file or from the command line. A property is referenced by enclosing the property name inside `${}`, for example `${basedir}`.
- **path**: A path is a set of directories or files. Paths can be defined once and referred to multiple times, easing the development and maintenance of build files. For example, a Java compilation task may use a path reference to determine the class path to use.

20.3 Built-in tasks

A comprehensive set of built in tasks are supplied with the Ant distribution. The tasks that we use in this example are described below:

ant	Invokes Ant using another build file
copy	Copies files and directories
delete	Deletes files and directories
echo	Outputs messages
jar	Creates Java archive files
javac	Compiles Java source
javadoc	Generates documentation from Java source
mkdir	Creates directories
tstamp	Sets properties containing date and time information
war	Creates WAR files

Restriction: Built-in task EAR is not supported in Ant 1.3 which is included in Application Developer. To create EARs, update Ant to 1.4 (as we mentioned, this causes it to be an unsupported environment), or use Ant extra tasks “Working with J2EE” on page 450.

20.4 Creating a simple build file

We created a simple build file that compile a Java source and put into a jar file. The build files are all named `build.xml`. This is the default name assumed by Ant if no build file name is supplied. This allows a build of the entire project or a single subproject to be performed by simply changing to the appropriate directory and issuing the `ant` command with no arguments. Our simple build file has the following common targets:

init Performs build initialization tasks—all other targets depend upon this target

compile Compiles Java source into .class files

dist Creates the deliverable jar for the module—depends upon the `compile` target

clean Removes all generated files—used to force a full build

Each Ant build file may have a default target. This target is executed if Ant is invoked on a build file and no target is supplied as a parameter. In all cases the default target for our build files is `dist`. The dependencies between targets are illustrated in Figure 20-1.

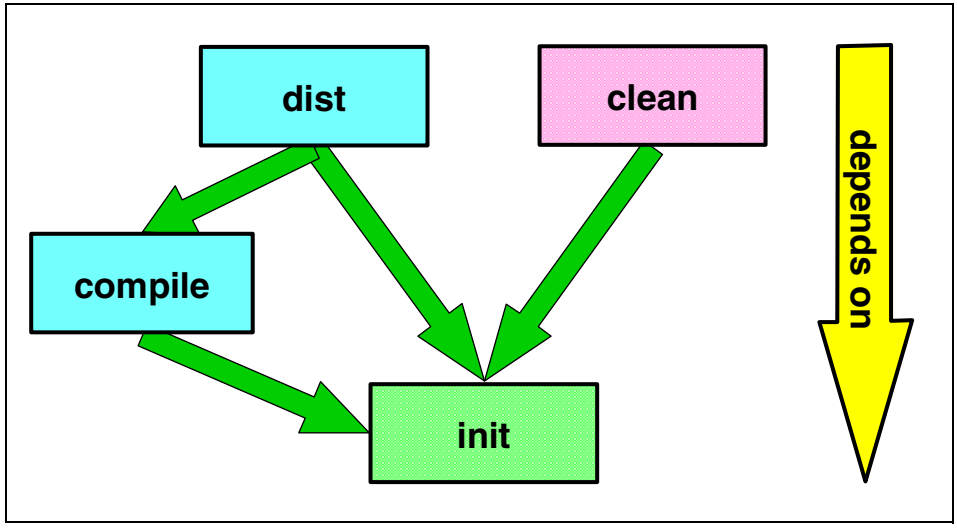


Figure 20-1 Dependencies

The following sample is a xml file to build simple Java class. The project tag is defining the project name and default target.

build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="SG246585-JAVA" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="src" value="."/>
  <property name="build" value="build"/>
  <property name="dist"  value="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init">

    <!-- javac resolver -->
   <available
      classname="org.eclipse.core.launcher.Main"
      property="build.compiler"
      value="org.eclipse.pde.internal.core.JDTCompilerAdapter"
      classpath="${java.class.path}"
   />

    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/SG246585-Java-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

20.4.1 Setting global properties

We use properties in our build file to provide the directory name to build and distribute. Those properties are used frequently in the target description. We defined the src property as ".". In Application Developer environment, the current directory is just under the project directory. To set the source in the Web project, we have to set the src property as "source". Properties also can be read from .properties file. Ant is able to read properties from files that use the format recognized by the Java `java.util.Properties` class.

```
<!-- set global properties for this build -->
  <property name="src" value="."/>
  <property name="build" value="build"/>
  <property name="dist"  value="dist"/>
```

20.4.2 Build targets

The build file contains four build targets.

Initialization target

The first target we describe is the `init` target. All other targets in the build file depend upon this target. In the `init` target we execute the `tstamp` task to set up properties that include timestamp information. These properties are available throughout the whole build. We also create a build directory defined as build property. This directory is used to keep the class files.

```
<target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>
```

Compilation target

The compile target is to compile all Java source which are located in the src folder, and the class file is saved in to build folder which is defined as destdir. We need to add the javac resolver using an available tag. This tag helps Ant to find the compiler. We can specify the compiler as an argument (we will discuss this later in 20.5, "Running Ant" on page 447).

```
<target name="compile" depends="init">

    <!-- javac resolver -->
   <available
      classname="org.eclipse.core.launcher.Main"
      property="build.compiler"
      value="org.eclipse.pde.internal.core.JDTCompilerAdapter"
```

```
      classpath="${java.class.path}"
    />

    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>
```

With this description, if the compiled code is the latest one, the source will not be recompiled, because the code is up-to-date. The following is the message in the Ant console (to see this message, -verbose argument should be set).

```
itso\wsad\sg246585\java\PartListApplication.java omitted as D:\WebSphere\Application
Developer\SG24-6585-JAVA\build\itso\wsad\sg246585\java\PartListApplication.class is up to date.
```

Dist target

Target dist creates a jar file which contains the projected files in the lib directory under the dist. The compile should be done successfully before the dist runs.

```
<target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/SG246585-Java-${DSTAMP}.jar" basedir="${build}"/>
  </target>
```

Cleanup targets

The last of our standard targets is the `clean` target. This target delegates responsibility to the `clean` targets.

```
<target name="clean">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
```

20.5 Running Ant

Ant is a built-in function and you can use it at anytime from a “Run Ant...” (Figure 20-2) context menu of any XML file. The Execute Ant Script dialog will open, showing the available Ant targets. You can check, in sequence, which ones are to be executed, and the execution sequence will be shown beside each target. You can also select Display execution log to Ant console, which will cause any Ant messages to be displayed in the Ant Console view (Figure 20-3). The view is opened when you run Ant, but to open the view manually, use the Perspective menu and use Show View -> Other -> Ant -> Ant Console.

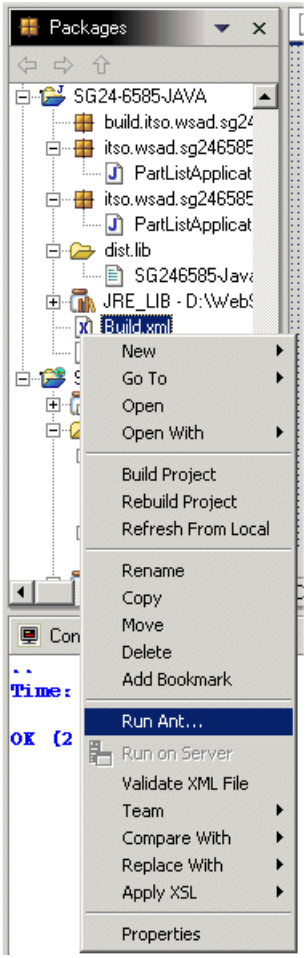


Figure 20-2 Run Ant

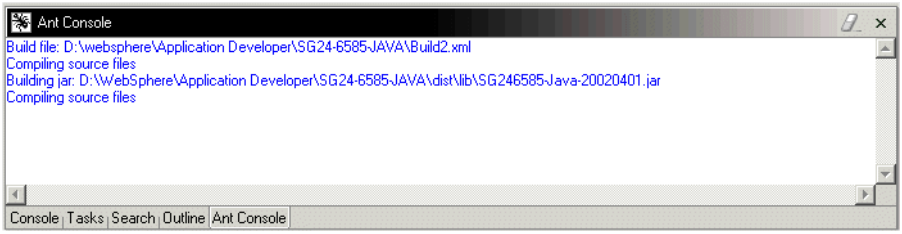


Figure 20-3 Ant Console

If you do not set the available tag in the build.xml, you get an error which is “Cannot use classic compiler” (Figure 20-4).

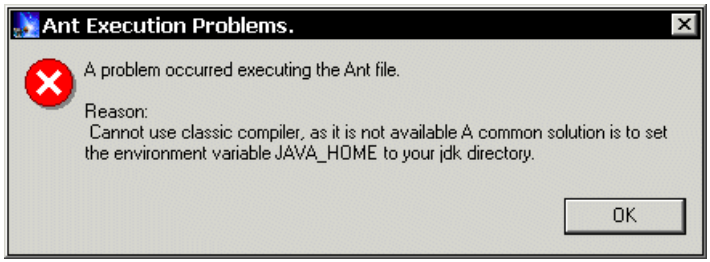

Ant Execution Problems.
A problem occurred executing the Ant file.
Reason:
Cannot use classic compiler, as it is not available A common solution is to set the environment variable JAVA_HOME to your jdk directory.
OK

Figure 20-4 “Cannot use classic compiler” error

Or you can use an argument to avoid this problem by specifying the following code in the argument input field (Figure 20-5).

```
-Dbuild.compiler=org.eclipse.pde.internal.core.JDTCompilerAdapter
```

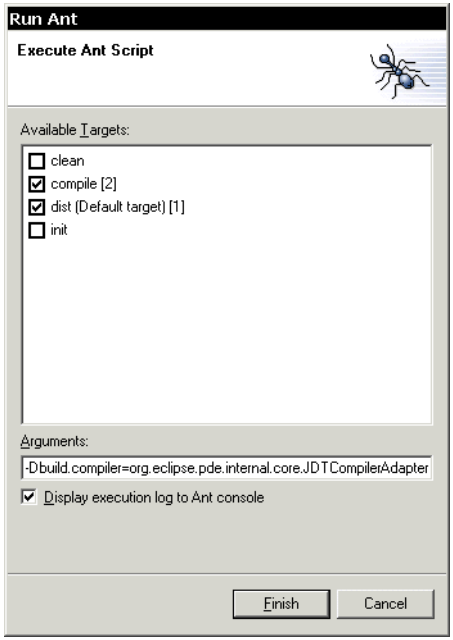


Figure 20-5 Run Ant dialog with -D argument

20.5.1 Classpath Problem

The classpath, specified in the property as Java build path, is not used in the Ant process. If one project is referencing the other project, the classpath must be specified as follows. In this case, it is referring to JUnit Java project.

```
<javac srcdir="${src}" destdir="${build}" includes="**/*.java">
   <classpath>
      <pathelement location="../JUnit"/>
   </classpath>
</javac>
```

20.6 Working with J2EE

During development, some of the deployment descriptor information is stored in ASCII XML files, but these files also contain some information in a format convenient for interactive testing and debugging. That is part of the reason it is convenient and quick to test J2EE applications on the internal WebSphere Application Server included with Application Developer. As part of the Application Developer functionality that creates standalone J2EE modules, this internally optimized deployment descriptor information is merged and changed into a standard format. The actual EAR being tested, and its WAR, EJB, and ClientApplication JARs, are not actually created as a standalone JAR file. Instead, a special EAR is used that simply points to the build contents of the various J2EE projects. Since these various individual projects can be anywhere on the development machine, absolute path references are used.

When an Enterprise Application project is exported, a true standalone EAR is created, including all the module WARs, EJB JARs, and Java utility JARs it contains. Hence, during the export operation, all absolute paths are changed into self-contained relative references within that EAR. To create a J2EE compliant WAR or EAR we need to use Application Developer export function, or create an AntTask to do so.

20.6.1 Ant Extra plug-in

What you need is a set of Application Developer Ant tasks to perform these build and export functions. Here is a plug-in to do these Ant tasks. In addition to the Ant tasks for the various J2EE export operations, it also includes Ant tasks to perform an EJB Deploy and to build Java utility JARs and include them within Ant builds of EARs. Specifically, these new Ant tasks are:

- **ejbDeploy**: Generates deployment code and RMIC code for an EJB Project.
- **ejbExport**: Exports an EJB project to an EJB JAR file.

- **warExport**: Exports a Web project to a WAR file.
- **appClientExport**: Exports an application client project to an application client JAR file.
- **utilJar**: JARs up a Java project into a JAR file within an enterprise application project.
- **earExport**: Exports an enterprise application project to an EAR file.

The original article and plug-in are available at following address:

http://www7b.boulder.ibm.com/wsdd/library/techarticles/0203_searle/searle1.html

To install the plug-in, close the Application Developer and copy it to the plug-in directory. Then restart the Application Developer and Ant Extra plug-in is ready by checking the preference dialog (Figure 20-6).

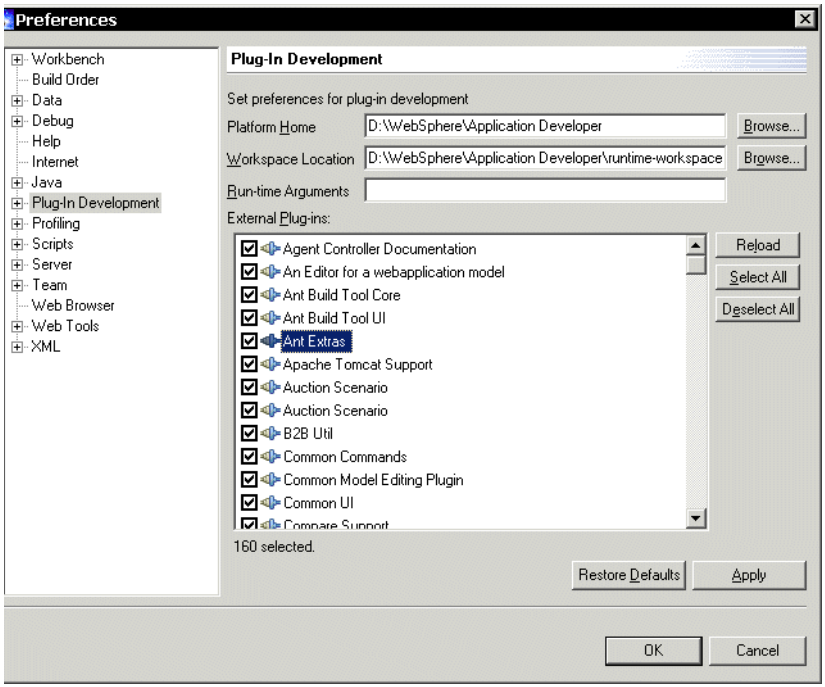


Figure 20-6 Ant Extra plug-in

20.6.2 Exporting a WAR file

The warExport tag is an extra ant task and it is enable to build a WAR file. The following code is the build.xml to build ItsoWsDealerWebProject.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project name="ItsoWsDealerWebProject" default="dist" basedir=".">

<target name="dist">
    <!-- Create the distribution directory -->
   <warExport WARProjectName="ItsoWsDealerWebProject"
              WARExportFile="c:\ItsoWsDealerWeb.war"/>
  </target>

</project>
```

This task not only create the war but also compile all need source codes. Following output is a result of this task.

```
beginTask null ...
null ... subtask: Building: /ItsoWsDealerWebProject. Invoking Lib Dir Builder on /ItsoWsDealerWebProject....
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:...
null ... subtask: Building: /ItsoWsDealerWebProject. Update ClassPath:...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:SERVERJDK_PLUGINDIR/jre/lib/rt.jar...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:/ItsoWsDealerWebProject/source...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:WAS_PLUGINDIR/lib/j2ee.jar...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:WAS_PLUGINDIR/lib/webcontainer.jar...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:WAS_PLUGINDIR/lib/ivjejb35.jar...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:WAS_PLUGINDIR/lib/websphere.jar...
null ... subtask: Building: /ItsoWsDealerWebProject. Catalog Lib Directory:DB2JAVA...
null ... subtask: Building: /ItsoWsDealerWebProject. Update ClassPath:...
null ... subtask: Building: /ItsoWsDealerWebProject. Set ClassPath:...
null ... subtask: Building: /ItsoWsDealerWebProject. ...
null ... subtask: Building: /ItsoWsDealerWebProject. Invoking Java Builder on /ItsoWsDealerWebProject....
null ... subtask: Building: /ItsoWsDealerWebProject. Reading resource change information for :
                                                                                ItsoWsDealerWebProject...
null ... subtask: Building: /ItsoWsDealerWebProject. ...
null ... subtask: Building: /ItsoWsDealerWebProject. Updating resources on the classpath...
null ... subtask: Building: /ItsoWsDealerWebProject. Build done....
null ... subtask: Building: /ItsoWsDealerWebProject. ...
null ... subtask: Building: /ItsoWsDealerWebProject. Java build completed...
null ... subtask: Building: /ItsoWsDealerWebProject. Invoking Table of Links on /ItsoWsDealerWebProject....
null ... subtask: Building: /ItsoWsDealerWebProject. ...
null ... subtask: Building: /ItsoWsDealerWebProject. Invoking Validation on /ItsoWsDealerWebProject....
null ... subtask: Building: /ItsoWsDealerWebProject. ...
null ... subtask: Building: /ItsoWsDealerWebProject. ...
null ... subtask: Updating....
```

Note: Both warExport and earExport do not support a variable in the file name. For example you CANNOT do [WARExportFile=”${build}/ItsoWsDealerWeb.war”].

20.6.3 Exporting an EAR file

The earExport tag is an extra ant task and it is enabled to build a EAR file. The following code is the build.xml to build ItsoWsDealerWebProjectEAR. With the default setting, source code is not exported in the EAR. To export the source, set ExportSource=”true”.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ItsoWsDealerWebProjectEAR" default="dist" basedir=".">
```

```
<target name="dist">
    <!-- Create the distribution directory -->
   <warExport EARProjectName="ItsoWsDealerWebProjectEAR"
              EARExportFile="c:\ItsoWsDealerWebEAR.ear"
              ExportSource="true"/>
  </target>

</project>
```

Part 6

Profiling

Part 6 introduces the Application Developer profiling tool and describes how to set up and run profiling.

© Copyright IBM Corp. 2002

21

Profiling concepts

This chapter discusses the following topics:

- Profiling architecture
- Performance analysis

We give an overview of the application profiling support in Application Developer, the type of problems it can be used to detect, and the different views of your application runtime profile that are provided. In the next chapter will look in detail about how to set up and run profiling.

© Copyright IBM Corp. 2002

21.1 Profiling architecture

Application Developer provides the developer with a set of tools to allow for early analysis of performance related issues in Java applications. The profiling tools can be used to gather performance information on applications that are running:

- Inside an application server like WebSphere
- As a standalone Java application
- On the same machine as Application Developer
- On a remote machine from Application Developer
- In multiple JVMs

Using filters you can focus on classes that you are interested in and omit tracing for others.

Traditionally performance profiling is something that is done once an application is getting close to deployment or even when it has already been deployed. Using the Application Developer profiling tools allows you to move this kind of analysis to a much earlier phase in the development cycle, therefore giving you more time to modify your architecture based on any problems detected.

Profiling creates a number of different graphical and tabular views of a Java program's run-time behavior, and facilitates identifying and diagnosing performance related problems.

The basic architecture of the profiling tools involves the JVM, (Java Virtual Machine), where the application is running, an agent running inside the JVM capturing profiling information, an agent controller that controls the agent and retrieves profiling information, and the performance analyzer inside Application Developer. The relationships between the components is shown Figure 21-1.

Note: To complete the performance analysis of your Web application you may also need to use tools such as WebSphere Studio Page Detailer to analyze the performance of the non-Java components of the application, (page sizes, number of bytes transferred, time to display a page and so on).

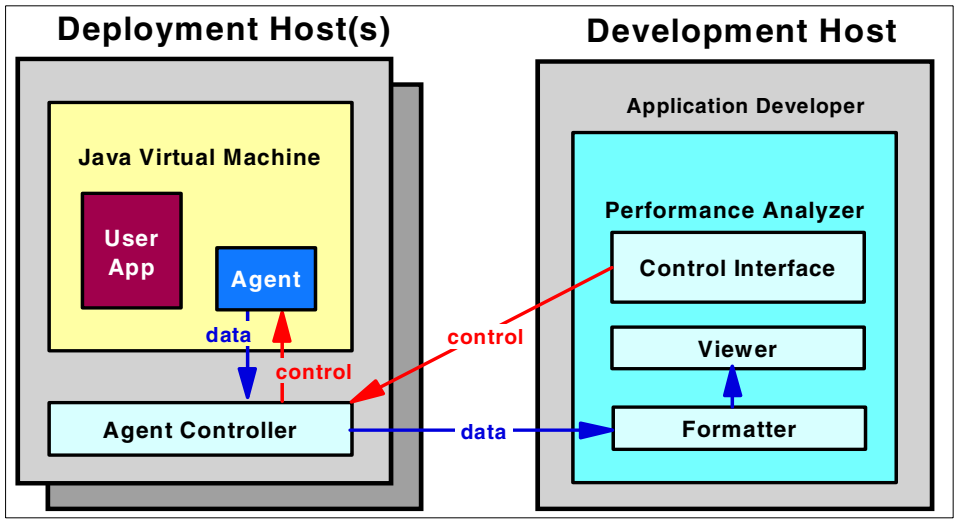


Figure 21-1 Application Developer profiling architecture

The agent runs inside the JVM and uses the Java Virtual Machine Profiler Interface, (JVMPI), to interface with the JVM. If you are interested in more information about JVMPI, the following Sun website has details:

`http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html`

21.2 Performance analysis

Using the performance analysis data gathered by the agent you can identify potential problems by focusing on:

- Time consuming classes and methods
- Memory intensive classes and methods
- Garbage collection statistics
- Objects that are not garbage collected
- Thread activity

To help you analyze the data returned by the profiler, Application Developer provides a number of different views that focus on different aspects of the data:

- Class statistics (tabular)
- Method statistics (tabular)
- Heap (graphical)
- Object reference (graphical)
- Execution flow (graphical)

The different views should be used in conjunction to build a complete picture of your application performance. This will provide you with the information you require to determine where you can most productively concentrate your efforts to improve performance.

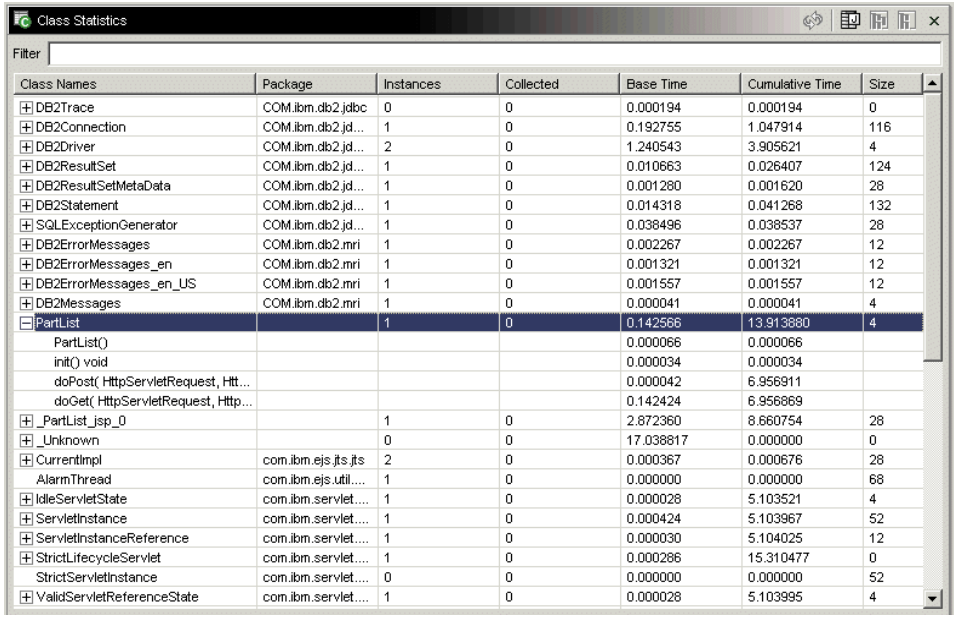
The views are linked together, that is if you have selected something in one view, the other views will show information about the same object. This makes it easy to collect all information about a particular object.

As an example, if you select a method from the Class Statistics view, you can switch to the Method execution view to get details about the execution of the method, or you can switch to the Object reference view to see what objects the method has created.

To update the information in the Profiling views to show the latest data captured by the profiler, select **Update Views** from the view menu. This will update all the views, not just the one you are currently in.

21.2.1 Class statistics

The Class Statistics view shows information about the classes in your application (Figure 21-2).


Class Statistics

Filter

Class Names	Package	Instances	Collected	Base Time	Cumulative Time	Size
DB2Trace	COM.ibm.db2.jdbc	0	0	0.000194	0.000194	0
DB2Connection	COM.ibm.db2.jd...	1	0	0.192755	1.047914	116
DB2Driver	COM.ibm.db2.jd...	2	0	1.240543	3.905621	4
DB2ResultSet	COM.ibm.db2.jd...	1	0	0.010663	0.026407	124
DB2ResultSetMetaData	COM.ibm.db2.jd...	1	0	0.001280	0.001620	28
DB2Statement	COM.ibm.db2.jd...	1	0	0.014318	0.041268	132
SQLExceptionGenerator	COM.ibm.db2.jd...	1	0	0.038496	0.038537	28
DB2ErrorMessages	COM.ibm.db2.mri	1	0	0.002267	0.002267	12
DB2ErrorMessages_en	COM.ibm.db2.mri	1	0	0.001321	0.001321	12
DB2ErrorMessages_en_US	COM.ibm.db2.mri	1	0	0.001557	0.001557	12
DB2Messages	COM.ibm.db2.mri	1	0	0.000041	0.000041	4
PartList		1	0	0.142566	13.913880	4
PartList()				0.000066	0.000066	
init() void				0.000034	0.000034	
doPost(HttpServletRequest, Htt...				0.000042	6.956911	
doGet(HttpServletRequest, Http...				0.142424	6.956869	
_PartList_jsp_0		1	0	2.872360	8.660754	28
_Unknown		0	0	17.038817	0.000000	0
CurrentImpl	com.ibm.ejs.jts.jts	2	0	0.000367	0.000676	28
AlarmThread	com.ibm.ejs.util....	1	0	0.000000	0.000000	68
IdleServletState	com.ibm.servlet....	1	0	0.000028	5.103521	4
ServletInstance	com.ibm.servlet....	1	0	0.000424	5.103967	52
ServletInstanceReference	com.ibm.servlet....	1	0	0.000030	5.104025	12
StrictLifecycleServlet	com.ibm.servlet....	1	0	0.000286	15.310477	0
StrictServletInstance	com.ibm.servlet....	0	0	0.000000	0.000000	52
ValidServletReferenceState	com.ibm.servlet....	1	0	0.000028	5.103995	4

Figure 21-2 Class Statistics view

By default this view shows the following information about the classes:

- Package to which a class belongs
- Number of object instances created using this class template
- Number of object instances of the class for which garbage collection occurred
- Total *base time* for all methods defined by the class
- Total *cumulative time* for all methods defined by the class
- Memory consumption by each object instance of this type

Base time of a method is the time spent executing this method only. It does not include time spent in other Java methods that this method calls.

Cumulative time of a method is the time the method spends on the execution stack, including both time spent in the method itself and in other methods that it calls.

Note: Times in profiling views are expressed in clock time, not CPU time. Be aware also that there is no compensation in the numbers for methods in a set that call each other, or for recursive calls. Some of the aggregated cumulative numbers may thus be inflated.

You can tailor the display by adding and deleting columns. Right-click anywhere in the view and select **Choose Columns** (Figure 21-3). You can add or remove columns from the view by selecting or deselecting them from the list.

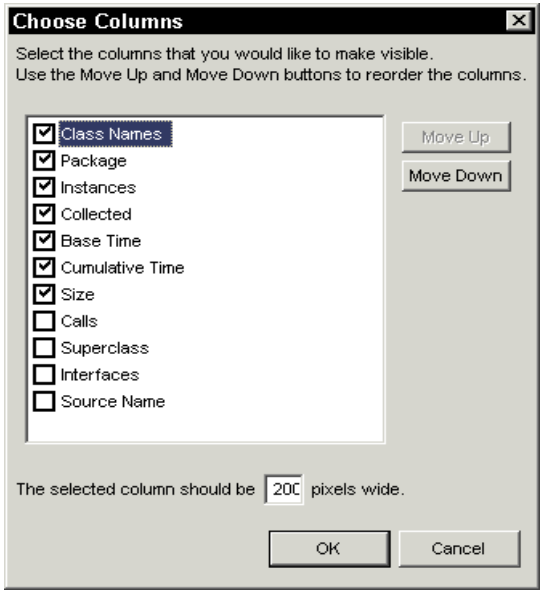


Figure 21-3 Choose columns dialog in Class Statistics view

You can sort on different columns by clicking in the column header. If you are interested in determining the most memory intensive classes, you would click in the header of the **Size** column. This would show the classes with the greatest memory footprint first in the view. Doing the same for the **Collected** column would show you the classes for which the most garbage collection has occurred.

21.2.2 Method statistics

The Methods Statistics view shows information about individual methods in your application. This view is essentially the same as the Class Statistics view but at a lower level (Figure 21-4).

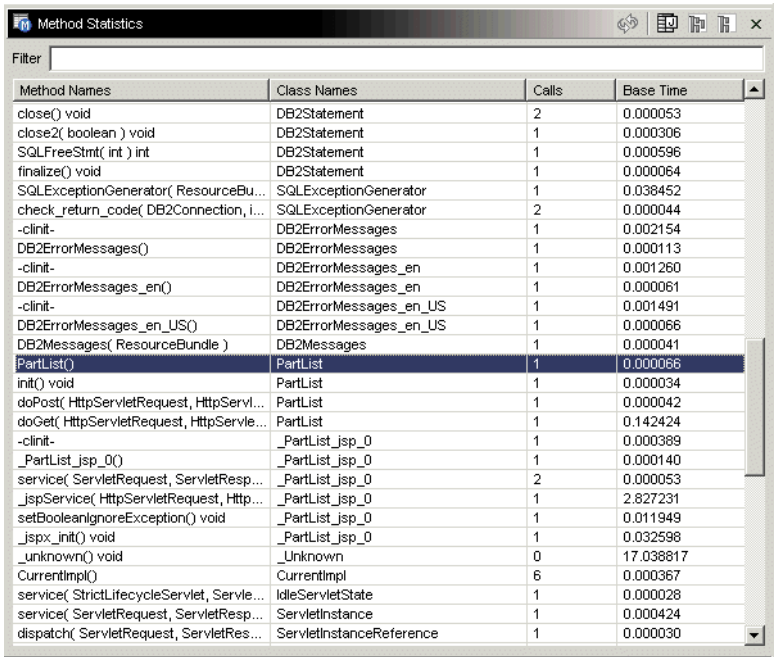

Method Statistics
Filter
Method Names
Class Names
Calls
Base Time
close() void DB2Statement 2 0.000053
close2(boolean) void DB2Statement 1 0.000306
SQLFreeStmt(int) int DB2Statement 1 0.000596
finalize() void DB2Statement 1 0.000064
SQLExceptionGenerator(ResourceBu... SQLExceptionGenerator 1 0.038452
check_return_code(DB2Connection, i... SQLExceptionGenerator 2 0.000044
-clinit- DB2ErrorMessages 1 0.002154
DB2ErrorMessages() DB2ErrorMessages 1 0.000113
-clinit- DB2ErrorMessages_en 1 0.001260
DB2ErrorMessages_en() DB2ErrorMessages_en 1 0.000061
-clinit- DB2ErrorMessages_en_US 1 0.001491
DB2ErrorMessages_en_US() DB2ErrorMessages_en_US 1 0.000066
DB2Messages(ResourceBundle) DB2Messages 1 0.000041
PartList() PartList 1 0.000066
init() void PartList 1 0.000034
doPost(HttpServletRequest, HttpServl... PartList 1 0.000042
doGet(HttpServletRequest, HttpServle... PartList 1 0.142424
-clinit- _PartList_jsp_0 1 0.000389
_PartList_jsp_0() _PartList_jsp_0 1 0.000140
service(ServletRequest, ServletResp... _PartList_jsp_0 2 0.000053
_jspService(HttpServletRequest, Http... _PartList_jsp_0 1 2.827231
setBooleanIgnoreException() void _PartList_jsp_0 1 0.011949
_jspx_init() void _PartList_jsp_0 1 0.032598
_unknown() void _Unknown 0 17.038817
CurrentImpl() CurrentImpl 6 0.000367
service(StrictLifecycleServlet, Servle... IdleServletState 1 0.000028
service(ServletRequest, ServletResp... ServletInstance 1 0.000424
dispatch(ServletRequest, ServletRes... ServletInstanceReference 1 0.000030

Figure 21-4 Method Statistics view

By default this view shows the following information about your methods:

- Method names
- Class to which each method belongs
- Number of calls made to each method
- Base time spent in each method

The **Cumulative Time** column time is not shown by default but can be added by right-clicking and selecting **Choose Columns**.

21.2.3 Heap

The Heap view is the most versatile profiling view and can be used to help you in a number of performance analyzing tasks such as:

- Identifying time-consuming objects and methods
- Identifying memory-intensive classes
- Gauging garbage collection
- Gauging program concurrency
- Identifying memory leaks
- Browsing method execution as a function of time.

The Heap view is color coded to make it easier to identify trouble spots in the code (Figure 21-5).

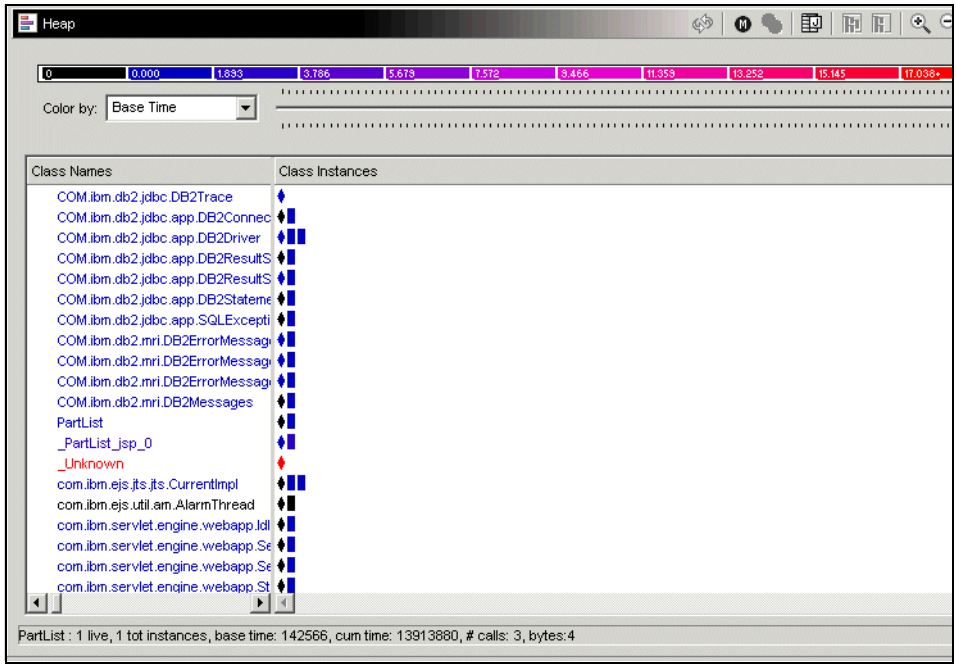


Figure 21-5 Heap view

This version of the Heap view shows base time per class. Red is used to indicate classes that have a relatively high base time. To switch to other types of information in this view, you select from the **Color by** drop-down combo box (Figure 21-6).

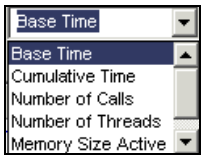


Figure 21-6 Heap view options

Each of the options will show a different perspective on the application performance. You can switch between **Object** and **Method** view using the icons in the toolbar at the top of the view, (and).

The status line at the bottom of the Heap view displays information about the currently selected object or method in the view.

The following sections describe how you can use the Heap view to help you with the tasks listed above. Note that the other profiling views can also be used for some of the tasks or to provide additional information.

Identifying time consuming objects and methods

1. From the Color scale selection panel above the view, select **Base Time** from the Color by field.
2. Move the slider, adjusting the color coding until you see a variation in the color of the rectangles that represent object instances, or diamonds that represent class objects in Objects mode; or the rectangles that represent the methods in Methods mode.
3. By default (that is, when the slider is positioned at the maximum value along the scale), you see mostly blue and black.
4. Select the object or method instance that is a color other than blue or black.
5. Observe the status line, and note the instance of the object or method. The status line also tells you the amount of base time that this instance consumed.
6. Select **Cumulative Time** from the Color by field, and repeat Steps 2 to 4.

Identifying memory-intensive classes

1. From the Color scale selection panel above the histogram, select **Memory Size Active** from the Color by field.
2. Move the slider, adjusting the color coding until you see a variation in the color for the bars that represent each class. By default (that is, when the slider is positioned at the maximum value along the scale), you see mostly blue and black.

3. Observe which classes have their corresponding bars in a color other than blue or black and have long bars in the histogram. Those classes consumed more active memory compared to the others.
4. You can change the **Color by** selection to *Memory Size Total*, and observe the methods that consume the greatest total memory.

For the Memory Size views (if you select either *Memory Size Total*, or *Memory Size Active* from the Color by field's list in the Color scale selection panel), a class name might be red, but the rectangle representing it in the histogram might be blue. The reason for this difference is that there are many instances of the class, but each one alone does not occupy much memory (blue). However, the total combination of all such instances takes up a lot of memory (indicated by the red font color for the class and the length of the bar in the histogram).

Gauging garbage collection

The objects for which garbage collection has occurred are represented by empty rectangles. By positioning the mouse over such objects, you get information (in the status line) on the time at which garbage collection occurred for the object.

Identifying memory leaks

1. View the histogram that is displayed in the Heap view in Objects mode.
2. Note whether there are a large number of color-filled rectangles. These rectangles denote instances for which garbage collection has not occurred.

Unexpectedly large numbers of such instances may suggest a memory leak. Source code analysis can help you to investigate these objects, and find their creators, and any objects that refer to them.

Browsing method execution as a function of time

1. Click the **Methods** button to switch to Methods mode.
2. Select a method in the Heap view. The status line displays the name of the method, the base time, the cumulative time, and the number of calls of that particular method. This information gives you an indication of the time (in seconds) that was spent executing that method.
3. Open the Method Invocation view by selecting **Show Method Invocation** from the method's pop-up menu. This view shows a representation of the same method as a labeled line. From the local toolbar, click the **Show Caller** button to see the callers of the selected method in the view. Each time you click this button, one method higher up in the calling sequence is displayed.
4. Select the method. The vertical length of the selected area indicates the base time for the method. You can determine the execution time for this method by

checking the vertical time scale on the right side of the view. The status line gives you the cumulative time for the method.

21.2.4 Object reference

The Object Reference view shows patterns of references in varying detail, both to and from a set of objects, (if such references exist). To display information, this view requires profiling data that contains object reference information. You can use this view to locate objects that still have references to them, study data structures, and find memory leaks. An example of an Object Reference is shown in Figure 21-7.

Note: To interpret this view you need to have a good understanding of how objects are allocated and referenced in your application and how Java handles object referencing.

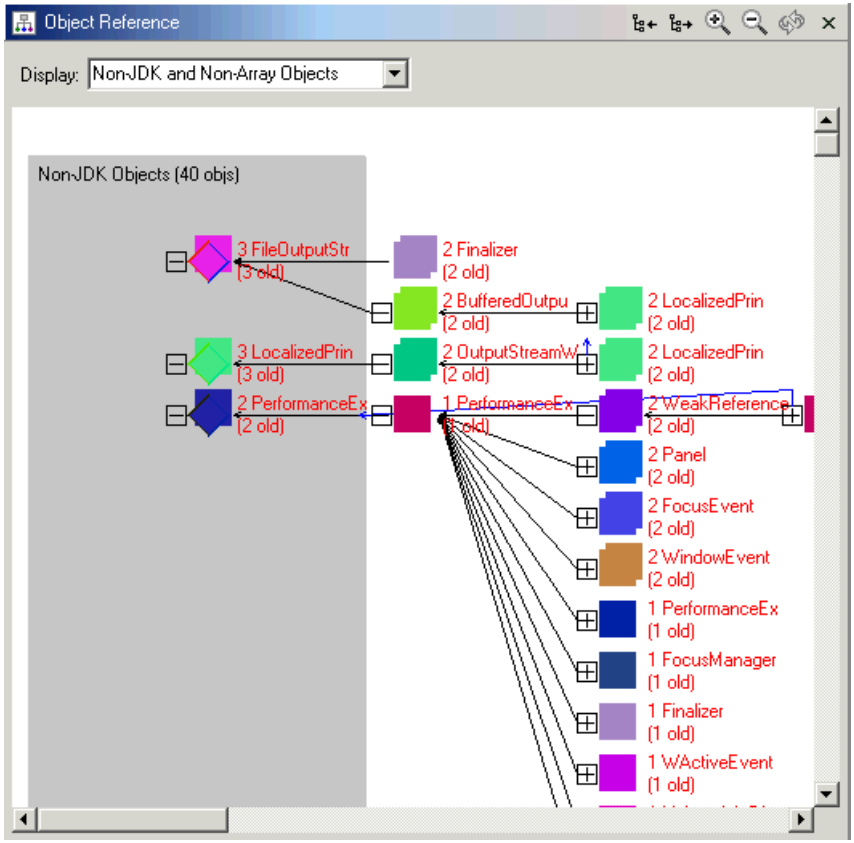


Figure 21-7 Object Reference view

This view displays the following information:

- **Base set of objects**: Appears in the gray area on the left. Each new object resides in this area once, but its instances also reside elsewhere. The view shows references to and from this base set of objects. You can select the base set of objects from the Display field.
- **Objects**: Represented by squares, each colored uniquely by class. Squares come in two forms:
  - A single square denotes a single instance
  - Twin squares represent multiple instances
- **Class objects**: Represented by diamonds.
- **References between instances**: Denoted by black arrows between the instances. An arrow points to the objects being referenced. Arrows point right to left.

- **Repetition of an object**: Represented by a blue, upward-pointing arrow. Click this arrow to display a long, blue arrow that leads to another place where the same object is displayed.
- **Old objects**: Denoted by a red font color for the number of objects and the class name, both of which appear to the right of each instance.

> **Note:** Old objects are those created prior to a certain point during the profiling session. You can specify that point as you create a dump of the object references using the **Dump Object Reference** icon in the toolbar. Objects that are created after that point in time are referred to as New-generation objects.

- **New-generation objects**: Denoted by a black font color for the number of objects and the class name, both of which appear to the right of each instance.

21.2.5 Execution flow

The difference between the Execution Flow view and the methods views, Method Execution or Method Invocation as described in “Browsing method execution as a function of time” on page 465, is that the Execution Flow view displays the execution of the entire program, while the method views display the execution of a single method at a time.

These three views are linked together. If you select something in the Execution Flow view and then switch to one of the method views, they will be displaying information for the method selected in the Execution Flow view. Also, if you have something selected in the Heap view, (a method, a class object, or a class instance), the Execution Flow view will have the same object selected when you switch to it.

You can use the Execution Flow view to identify active threads, long-lived or frequently called methods, and phases of program execution. You can select **Zoom In** from the view menu to focus on specific parts of the graph and **Zoom Out** to see the bigger picture.

When you are in the Execution Flow view, or in the Method Invocation or Executions views, you can open the editor on a method by right-clicking on it and selecting **Open Source**. This is true for classes whose source files exist in the workbench.

An example of an Execution Flow view is shown in Figure 21-8. When you move the cursor over the vertical bars, the status line at the bottom of the view will show the method name and other related information.

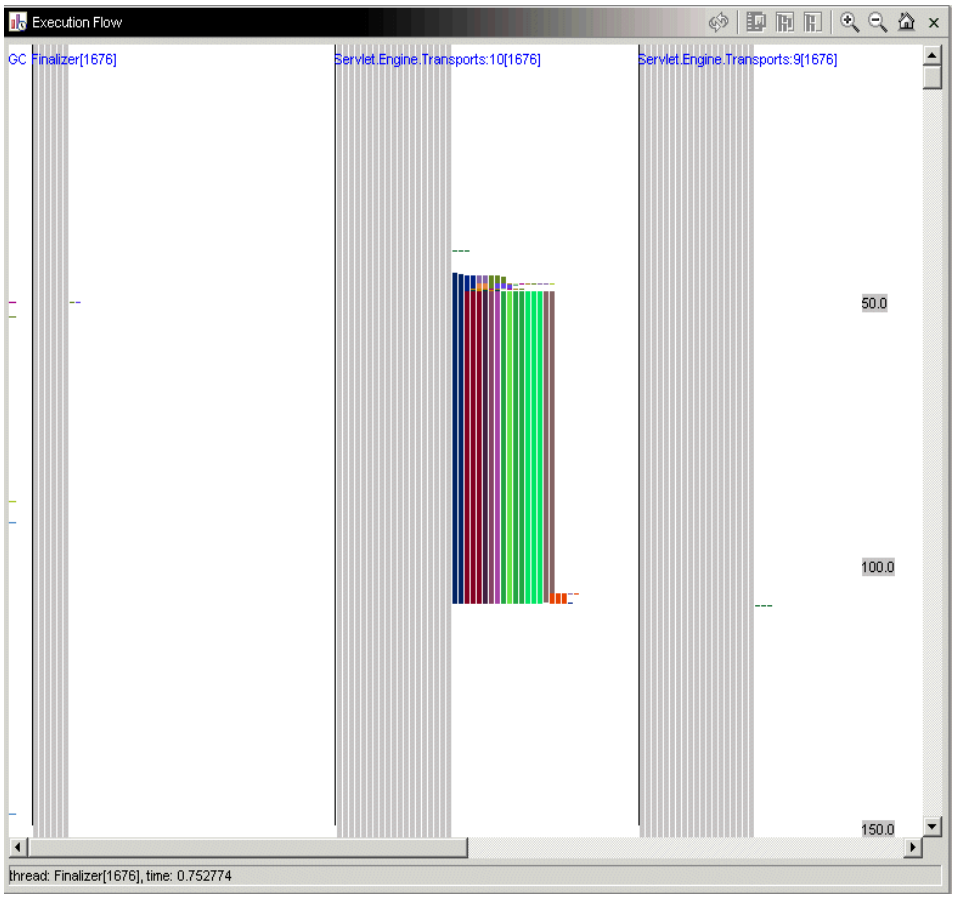


Figure 21-8 Execution Flow view

Horizontally you can see the active threads, with the garbage collection thread, (GC), always shown at the far left. At the far right of the graph is the time scale showing elapsed time in seconds.

Each vertical bar represents a method and the height of the bar represents the time of execution for that method. The color of the bar identifies the class of the object, that is bars of the same color represents methods belonging to the same class.

You can use the **Zoom In** action on the view menu to show parts of the graph in higher resolution. As you do this, you will see the time scale at the right of the graph get more granular. If you want to see more details about a method's execution, select it in this view and then switch to the Module Execution view.

To reduce the amount of information in the view, you can remove threads from it. To do this select **Threads** from the view menu and deselect any threads that you are not interested in at the moment. From the menu you can also show or hide all repetitions of methods.

22

Profiling your application

This chapter discusses the following topics:

- Configuring WebSphere Test Environment for profiling
- Starting and stopping profiling
- Profiling remotely using the Remote Agent Controller

© Copyright IBM Corp. 2002

22.1 Configuring WebSphere Test Environment

The normal environment for profiling during development is the built-in WebSphere Test Environment in Application Developer. Before you can start using the profiling tools, you need to make some changes to the default configuration of your server instance:

- Enabling the profiling agent
- Turning off the JIT compiler

22.2 Enabling the profiling agent

You need to ensure that the profiling agent is enabled on the server instance where you application is running. Switch to the Server perspective and find the Server instance in the Server Configuration view. Double-click it to open its editor (Figure 22-1).

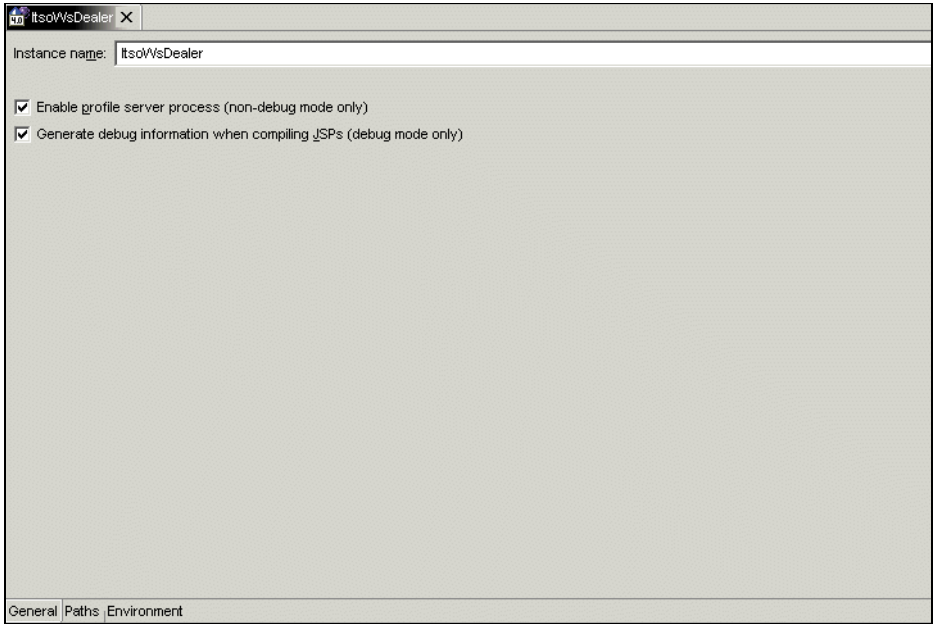


Figure 22-1 Enable profiling agent

To enable profiling ensure that the **Enable profile server process** check box is selected.

You also have to disable the Just-in-time Java Compiler. This is necessary to ensure, that all code will run through the JVM and also will be available to the profiling agent. To do this switch to the **Environment** tab on the Server Instance configuration editor (Figure 22-2).

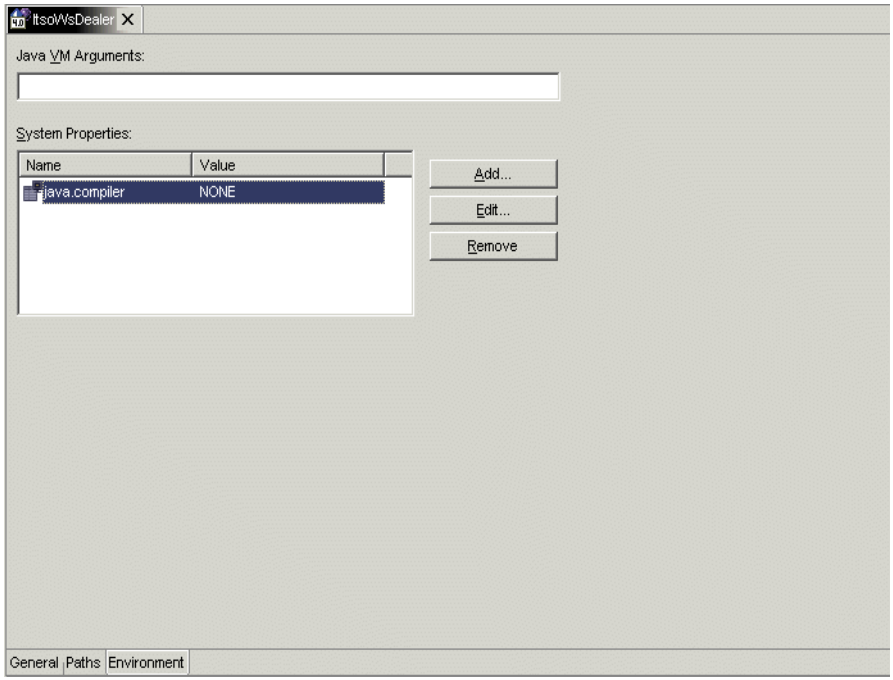


Figure 22-2 Disable the JIT compiler for profiling

Click **Add** and define a new system property as shown in the figure. Close and save the view.

Important: Once you have made the necessary modifications to the server instance properties, you need to restart the server to make the changes take effect.

In the console output from the server startup you should see a message similar to this:

```
WebSphere AEs 4.0.2 a0150.05 running with process name localhost/Default
Server and process id 636
Host Operating System is Windows 2000, version 5.0
Java version = J2RE 1.3.0 IBM build cn130-20010925was (JIT disabled), Java
Compiler = NONE
```

Important: You should make a note of the **process id** since you will need to know it when you start the profiling.

You should also verify that the Agent Controller is started on the host where the application is running. On Windows you can do this by clicking the **Start** button and selecting **Settings->Control Panel**. Then select **Administrative tools** and **Services**. You should see the following entry in the list of services (Figure 22-3):



Figure 22-3 IBM Agent Controller service entry

On UNIX, the Agent Controller runs as a daemon.

Note: The Agent Controller is installed automatically along with Application Developer, but if your application is running on a non-Application Developer machine it has to be installed separately.

22.3 Starting and stopping profiling

When you have configured and started your server instance, you are ready to start profiling your application.

The profiling is controlled from the Profiling perspective. To open the perspective select **Perspective—>Open—>Other** and choose **Profiling** from the list (Figure 22-4).

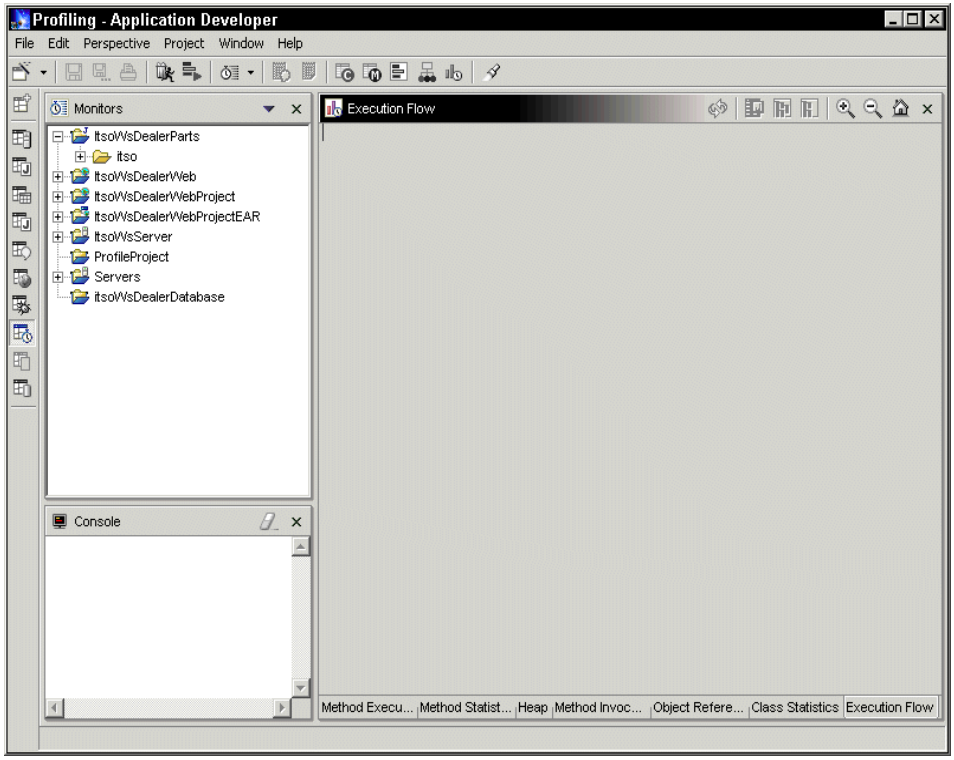


Figure 22-4 Profiling perspective

The right pane is where your profiling views will be shown once you have started monitoring. In the top left pane you can see an empty node called **ProfileProject**. This is where your profiling sessions will appear.

To create a new profiling session select the profiling icon  from the toolbar. Use the dropdown menu to select **Attach—>Java process** (Figure 22-5).



Figure 22-5 Attach to a running process for profiling

We want to run the profiling against the process that is already running, namely the server process that we started earlier. From this menu you could also launch a new process.

In the dialog that is displayed you should see the process id of the server process listed (Figure 22-6).

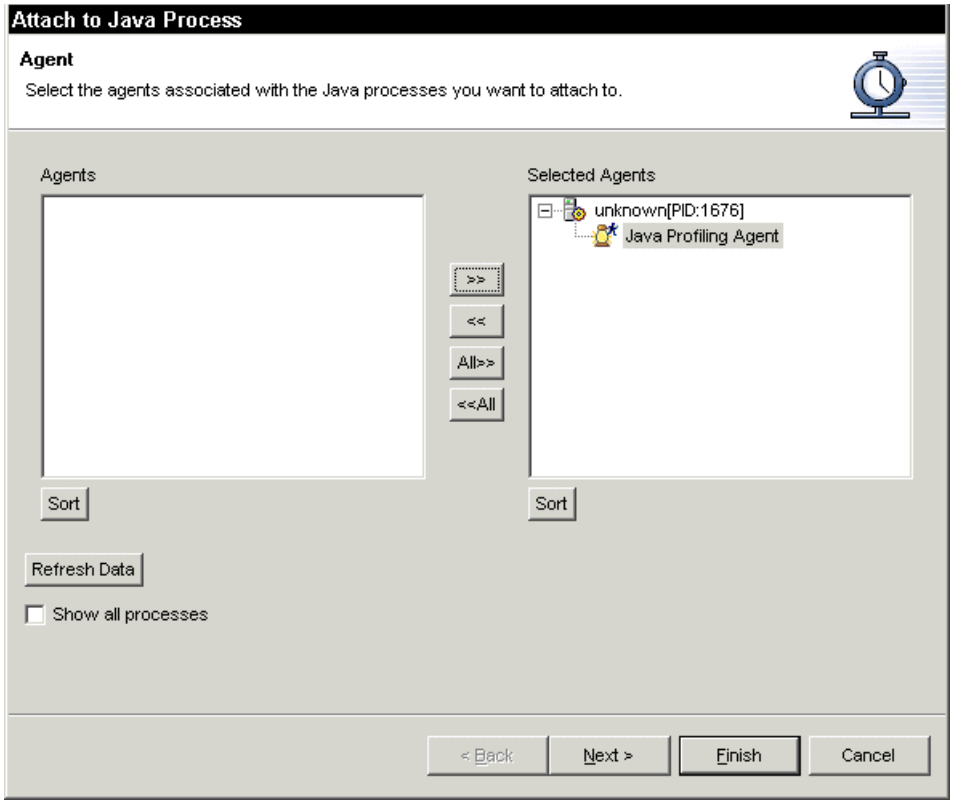


Figure 22-6 Attach Java process - select agent

Move the agent across to the Selected Agents pane by clicking **>>** and then click **Next**. On the second page you can change the default name of the monitor. Normally you will leave the defaults on this page (Figure 22-7).

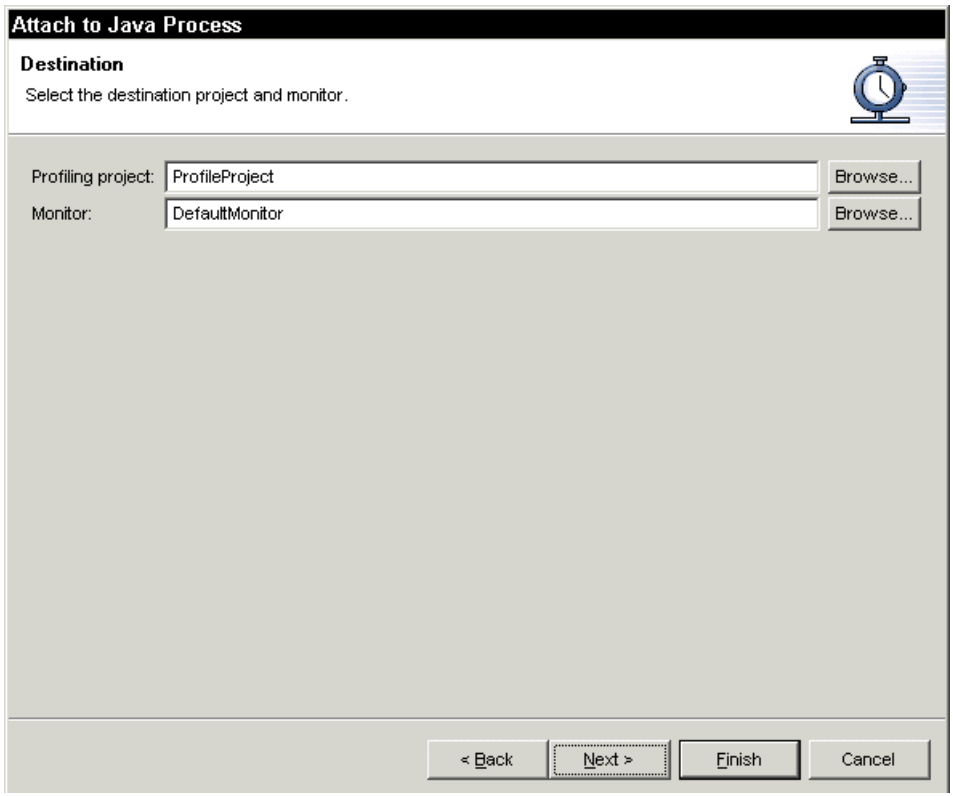


Figure 22-7 Attach Java process - select project and monitor

On the final page of the wizard you can modify the default filters (Figure 22-8).

Normally you don't want to include system classes in the profiling. If you for some reason do, you can disable one or more of the default filters by deselecting the **Enabled** check box next to the filter name.

You can add new filters for other classes that you may want to exclude by typing in the name in the **New Filter** entry field and clicking **Add Filter**.

The order of the filters can be changed by using the **Move Up** and **Move Down** buttons. The order is important when you want to profile a specific sub-package, but not the parent package. As an example, if you wanted to profile `com.ibm.myprogram` you have to `INCLUDE` that before you `EXCLUDE com.ibm.*`.

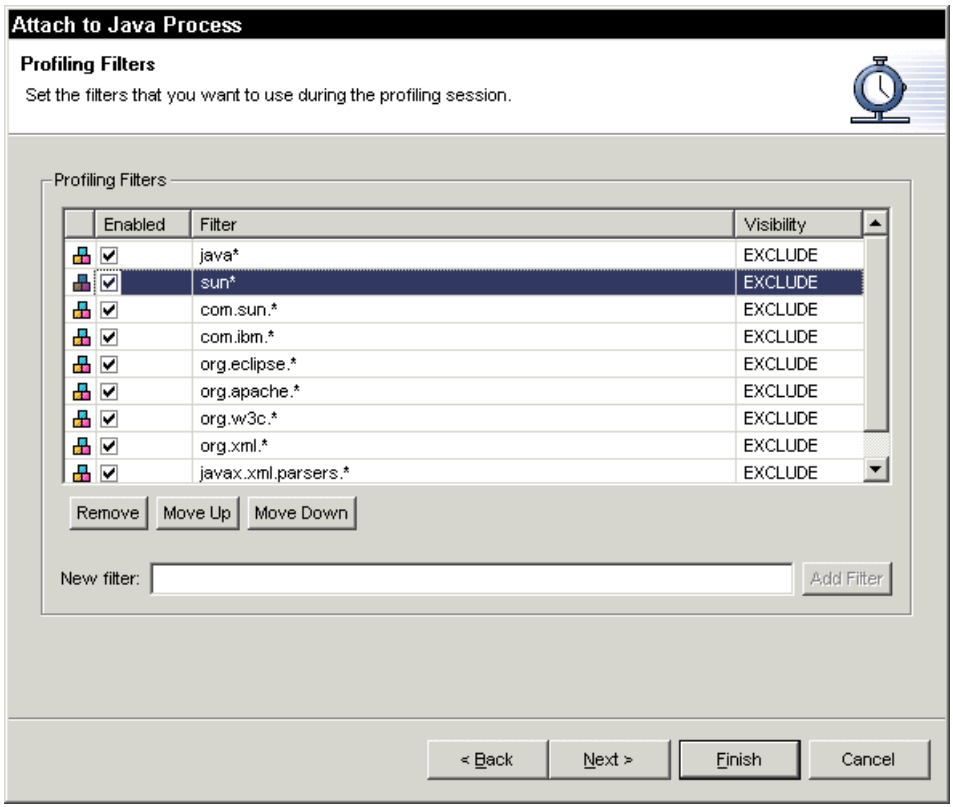


Figure 22-8 Attach Java process - set filters

When you click **Finish** an entry will be added under ProfileProject.

To start monitoring, right-click the Profiling object under the project and select **Start Monitoring**. To access the different profiling views, use the icons on the toolbar:

-  for **Classes** view
-  for **Methods** view
-  for **Heap** view
-  for **Object References** view
-  for **Execution** view.

22.4 Profiling remote processes

The profiling feature in Application Developer allows you to monitor applications running on a remote host. The only requirement is that the host has to have the IBM Agent Controller installed and running. This will be the case if the remote machine has Application Developer installed.

If Application Developer is not installed on the remote host, the Agent Controller has to be installed separately. IBM Agent Controller is available for many different environments, including AIX, HP, Windows, zOS, i-series, Linux and Solaris.

Attaching a Java process on a remote machine is done in the same way as described in 22.3, “Starting and stopping profiling” on page 474, for a local application. There is one extra page in the wizard where you need to specify the network address of the remote computer (Figure 22-9). The rest of the wizard pages are the same as for setting up a local profiling session.

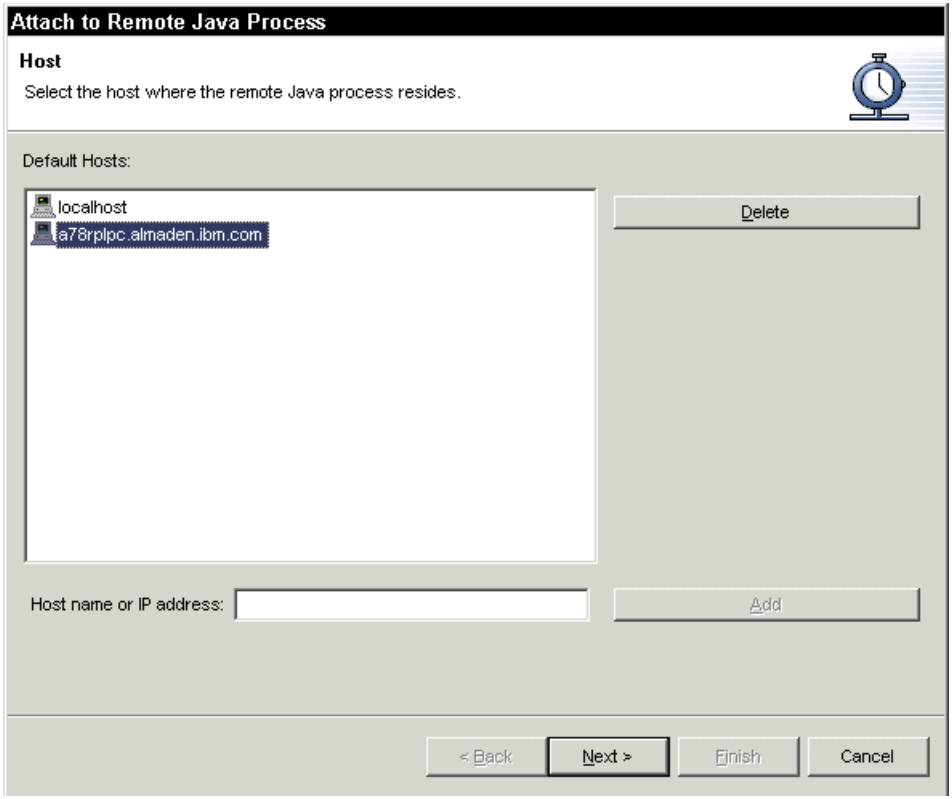


Figure 22-9 Attach Java process - select remote host

22.5 Some things to be aware of

- If the JVM you are profiling is external from Application Developer you may need to add the Remote Agent Controller bin directory, (normally `C:\Program Files\IBM\IBM Agent Controller\bin`), to the windows `PATH` to allow the JVM to locate the agent `DLL`.
- Because communications to and from a Remote Agent Controller is done via TCP/IP sockets, firewalls may block the communication.
- There is currently no authorization in the Remote Agent Controller, so it's up to you to make sure that only authorized users can access it.
- Views have to be refreshed to show latest data by using **Update Views** from any profiling view.
- You can *not* use profiling and debugging at the same time.

Part 7

Team programming

Part 7 introduces the Application Developer version control mechanism and team development using CVS and Rational ClearCaseLT.

© Copyright IBM Corp. 2002

23

Version control

Application Developer is a file based, integrated development environment (IDE). It maintains a local *workspace* where all the project data is stored in files. For a developer who is working “off-line”, that is not as a part of a team, a local history of changes is maintained within the workspace. The history of resource changes is cached, allowing the developer to compare and replace resources with earlier versions.

In most cases a developer does not work alone but as a part of a team. This is when the issue of team development management becomes crucial. Application Developer provides an open architecture that provides plug-in support for different Software Configuration Management systems (SCMs).

This chapter will discuss the following topics:

- Workspace
- Standalone and team development
- Team development terminology
- SCM integration with Application Developer

© Copyright IBM Corp. 2002

23.1 Workspace

Application Developer maintains a *workspace* where the project data (files) are stored. Every time you save a resource, the changes are stored in the file system. Normally this is your local file system, but you can also choose to store your workspace directory on a distributed file system. The installation default is to use *C:*\<Application Developer_ROOT>*workspace* to hold project resources.

The workspace directory can be specified when Application Developer is started by using the -data workspacedirectory call parameter.

Tip: If for some reason you ever need to start over from scratch, the quickest way to achieve this is to exit Application Developer and simply delete all the directories and files below the workspace directory. When you then restart Application Developer you will have a clean workspace.

Caution: All the changes, that were not saved beforehand (exported or released) will be lost!

23.1.1 Working with the local history

The Application Developer IDE has the following features:

- All deletes are permanent. There is no recycle bin.
- A history of all changes is maintained locally and files can be reset to a previous state.

The local history of each file is updated when you create or modify a file. Each time you save a file, a copy of its previous state is also saved.

This allows you to compare the current file state to a previous state and to replace the file with a previous state. Each state in the local history is identified by the date and time the file was saved.

Note: Only files have local histories, not projects or folders.

Figure 23-1 is an example of what the local history for a workbench file might look like:

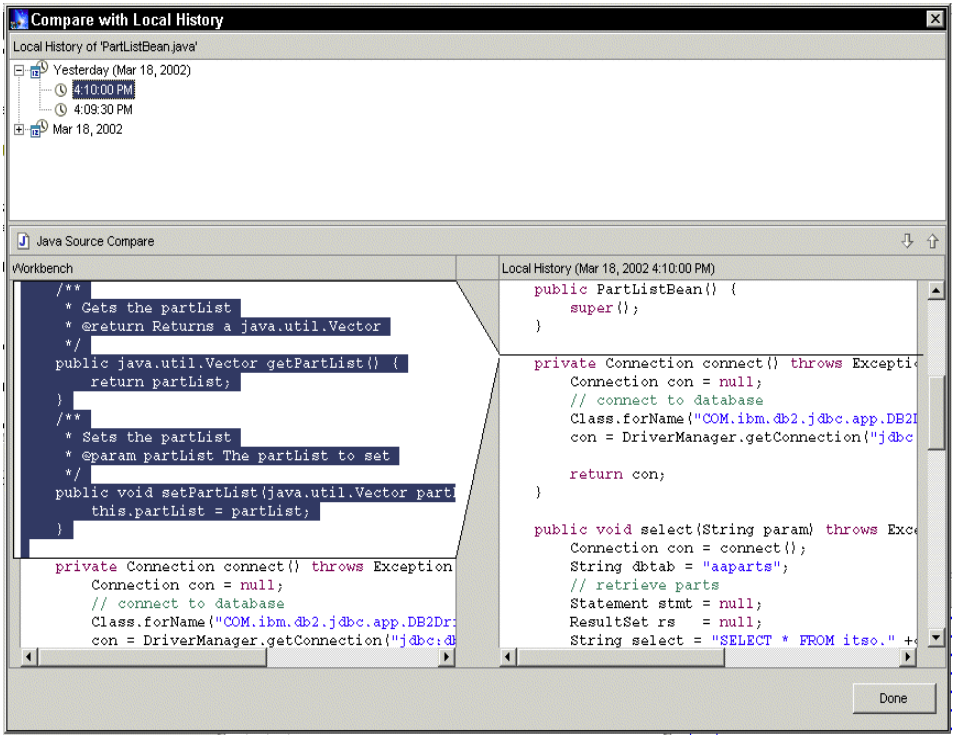


Figure 23-1 Compare with local history window

Application Developer allows you to compare and replace your current code with one of its locally managed versions.

To view the local history of a file, choose **Compare with—>Local History** from its context menu. You can select different local states in the list, which are then compared against the current version of the file (Figure 23-2).

You can also revert to the previous state of a file by selecting its **Replace With—>Local History** menu item.

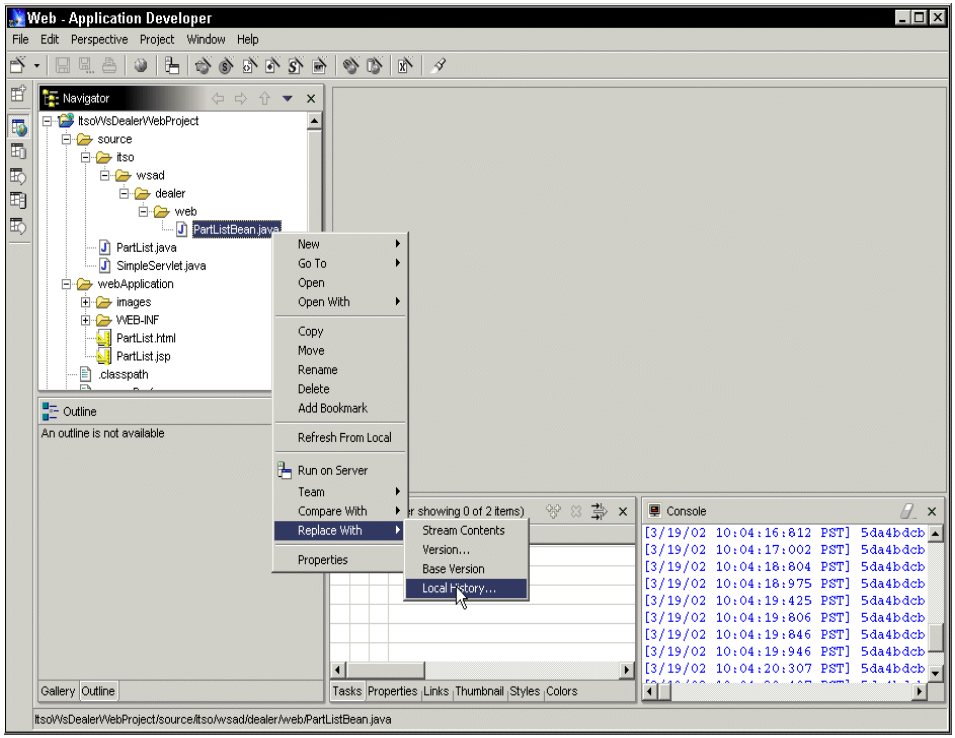


Figure 23-2 Starting compare/replace with local history

Note: These local histories are independent of the team development environment and the shared team repository.

You can set the amount of disk space along with other history attributes for Application Developer to use for local history information on the Workbench Preferences page (**Window—>Preferences—>Workbench—> Local History**).

23.1.2 Configuring multiple workspaces

Since there is a memory overhead associated with having many open projects in the workspace at the same time, you may want to configure multiple workspaces for the different development projects you are working on.

This reduces the amount of “clutter” in the Navigator view, and also minimizes the number of entries in the Server control panel, make it easier to find a particular server configuration or instance.

Each workspace is created as a separate folder in the file system. To start the IDE with a different workspace, simply execute:

```
wsappdev -data myworkspacedir
```

Where, `myworkspacedir` is the absolute path to the workspace folder to be used for this session. If this is the first time the workspace has been used, the IDE displays the default view specified when installing Application Developer.

If you have sufficient memory available on your workstation, it is possible to execute multiple instances of the IDE simultaneously against different workspaces. This technique is useful for example when working on two releases of a project at the same time, such as building a new release and performing maintenance on an old one. Each development stream can have its own workspace and copies of the project files.

If a workspace is not explicitly defined when starting the IDE, the default workspace provided in the installation directory will be used.

Tip: To enable versioning of your project data we recommend you that you use a versioning system even when you are planning to work in a single workstation environment.

23.2 Team development

To enable team development you need to have a *shared repository*. This repository is where all the development artifacts will be stored, maintained, and shared between a team of developers.

The team development environment in Application Developer enables use of a pluggable repository and implements an *optimistic concurrency model*.

A repository is a persistent data store that coordinates multi-user access to software versions (projects) and software snapshots (team streams).

Version control systems provide two important features required for working in a team environment:

- A history of the work performed by the team members
- A way to coordinate and integrate this work

Maintaining a history is important in order to track changes by comparing current working products against earlier versions, and to be able to revert back to a previous version if required.

Coordinating the development work is essential to ensure that only one copy of the current project state exists at any specific point in time. This copy will contain the integrated work of the whole team. This coordination is provided via the *stream model*.

23.2.1 Team roles

Important: Unlike some other repositories, the workbench does not have any notion of *object ownership*. Write access is controlled only by the write permission to the directory that holds the resource.

Anyone who is authorized to access the resource can create, edit, or delete it.

Typically each SCM user will have an account on the server so that actions can be logged to a particular user. The creation of users and groups is done by the SCM server administrator.

Important: Unlike Visual Age for Java there is no restriction that a package can be owned by only one project. This gives you more flexibility in designing your project structure.

23.2.2 Team terminology

The Application Developer team development environment uses the following terminology.

Repository

A repository is a persistent store that coordinates multi-user access to projects and their contents. Projects in a repository can be of two forms: *immutable* (a project version) or *modifiable* (project in a stream). Communication between the repository and workbench clients is possible over both local and wide area networks.

Stream

A stream is a shared workspace on a team server where project data is stored. Teams share and integrate their ongoing work in streams.

Think of a stream as a shared work area that can be updated at any time by a team member. In this way individuals can work on a team project, share their work with others on the team, and access the work of other developers during all stages of the project.

Note: The stream effectively represents the current shared state of the project.

Resources can be changed in the team members' individual workspaces without affecting the stream. Each developer must explicitly release changed resources to the stream.

Every repository has at least one stream. Under certain conditions, more than one stream can exist in a repository.

There are several ways you could arrange your team project structure. For example, your team could treat the **HEAD** stream as the released stream and have a second stream for development, or you could have one release stream per department or even one for each developer.

Projects may be developed in multiple parallel streams. You could for example be making fixes to Version 1.3 of a product in order to produce Version 1.31, while at the same time working on the development of Version 2.

Synchronizing

While you are working on the project on your workstation, all changes are local. When you are ready to make your local resource changes available to other team members, you must *release* your work to the stream. Releasing means making your changes available to the team, that is copying changed files to the team stream. All such changes will be marked as *outgoing changes* when you perform the synchronization.

Catch up

Catching up is the process of copying changes other developers have made to a resource into your local workspace. This ensures that you will have the latest work from the other team members incorporated in your work as well.

Important: Before releasing to a stream you should catch up to it.

Release

After you have caught up with the stream, merged any conflicting changes in your local workbench, and tested all changes locally, you can safely release your changes to the stream. In this way you can be sure that you are not overwriting work that has been done by other team members.

When you release your changes to the stream, they will be copied from your local workspace to the stream. As a result, these changes will be seen as *incoming changes* when other developers later catch up to the stream.

Versioning

Resources can be versioned at any time in order to capture a snapshot of the current state of the resources at a specific point in time. Versions constitute the baseline (“frozen code”) of a project at particular times.

Version

When a resource is versioned, a non-modifiable copy of it is released to the repository. If you change a resource in your workspace after synchronizing, then you have modified the resource.

It is possible to version a file or folder without versioning the project that contains it by synchronizing it with the stream. However, there is no need to explicitly version resources other than projects. During the synchronization of the whole project these types of resources will be implicitly versioned when they are released to the stream. In other words, versioning a project saves the set of all resource versions in that project.

The difference between versioning from the workspace or from the stream is in deciding which child resource versions should be part of the project version.

When you *version a project from the workspace*, the base version of the resources in your workspace will be captured as part of the project version. This is the preferred method of versioning a project because when doing so you know exactly which resource versions will be released into the project version.

When you *version a project from the stream* you include the latest resources versions from the stream as part of the new version of the project. You should not version your projects from the stream if you are not sure what is currently released in the stream.

Base Version

The base version of a resource is what was released to the stream at the time of synchronization.

Therefore the base version of a resource in a workspace denotes its corresponding state in the repository. This remains true even after you modify the resource in your workspace. The base version shows what the resource looked like before you started modifying it. This allows a developer to determine what modifications have been made to a resource by comparing it to its base version.

Important: Projects do not have a base version.

Local history
A history of all files that you have changed is cached locally. This allows you to compare a resource with a previous version and, if required, replace it with that version.

The local history of each file is maintained when you create or modify the file.

Attention: The local history of a resource will be reset every time you synchronize with the stream.

A new history for a resource is created the first time you modify the base version of it.

Enabling the **Show Version Info** menu item in the Navigator or Packages view displays the base version of each released file.

23.2.3 Optimistic concurrency model

Resources normally don't exist in isolation but typically have implicit or explicit dependencies on other resources. For example, Web pages have links to other Web pages, and source code has references to artifacts defined in other source code resources.

As resources are released to the stream, these dependencies can be affected. Ensuring the integrity of the dependencies is important because the stream represents the current valid project state. At any point a team member should be able to take the stream contents as a basis for new work.

An *optimistic concurrency* model is one in which any member of the team can make changes to any resource that they have write access to (Figure 23-3). The model is termed *optimistic* because it assumes that conflicts are rare. This corresponds to the separation of work in most development projects, whereby each developer has the responsibility for a set of resources.

However, because there is always the potential for two team members to release changes for the same resource in the repository, the development process should be designed to detect and deal with such conflicts if and when they occur.

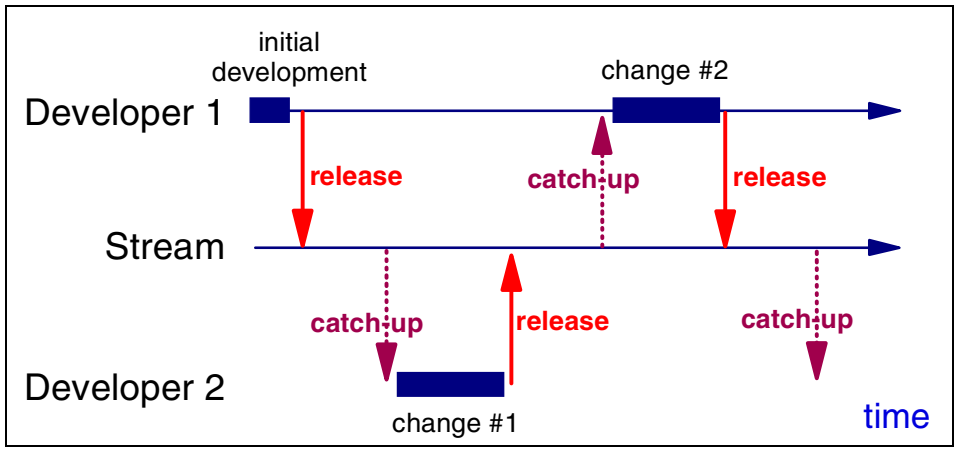


Figure 23-3 Optimistic concurrency model - sequential development scenario

23.2.4 Ideal work flow

The ideal work flow is one in which the stream integrity is always preserved and change conflicts are detected and dealt with. Such a work flow should follow these steps:

1. **Start fresh**: Before starting to work, the developer should catch up with the current stream state. If there is no local work that needs to be preserved, the fastest way to achieve this is to select the project or projects from the stream and select **Add to Workspace**. This will overwrite any local copies of project resources with those from the stream.
2. **Make changes**: The developer can now work locally in the workspace. New resources can be created and existing ones modified. All changes are saved locally in the workspace file system.
3. **Synchronize**: When the developer is ready to release the new and updated resources, the work needs to be synchronized with the stream.
4. **Catch up**: The developer can examine all incoming changes and add them to the local workspace. In this way it can be determined whether there are changes which might affect the integrity of what is about to be released. The developer needs to resolve conflicts, re-test, then run integrity checks (for example check for broken hypertext links, ensure the code compiles, and so on).
5. **Release**: When the developer is confident that the changes are well integrated with the latest stream contents, they can be released to the stream. To be absolutely sure, Step 4 might be repeated before the final release to check that there are no new incoming changes.

Under some circumstances this work flow may be simplified. You may be confident that incoming changes do not affect you, and choose to release without catching up. In general, however, team members should be encouraged to follow a flow similar to this to ensure that the stream integrity is not accidentally compromised.

23.2.5 SCM integration

The team development model of the workbench enables the use of pluggable third party repositories rather than mandating the use of a proprietary repository.

A repository is a persistent data store that coordinates multi-user access to software versions (projects) and software snapshots (team streams).

In comparison with a proprietary approach, (for example the one used in VisualAge), a file based system offers the following advantages:

- Easier integration of other tools since you are working directly in the file system
- More flexibility in source management and team development

In the workbench, developers do all of their work in individual *workspaces.* Periodically they make changes to the *team stream* in the software configuration management system (SCM) that has been installed.

This model allows individual developers to work on team projects, share their work with others as changes are made, and access the work of other developers as the project evolves.

At any time, developers can decide to update their workbench by retrieving changes that have been released to the team stream by other team members (referred to as "*catching up*"). They can submit their own changes to the shared team stream to enable other team members to access them (referred to as "*releasing*").

Communication between the repository server and workbench clients is via TCP, which enables development over local as well as wide area networks.

Pluggable SCM architecture

The version and configuration management architecture of the workbench platform (Figure 23-4) enables different vendors to integrate their SCM repositories through the workbench SCM adapters.

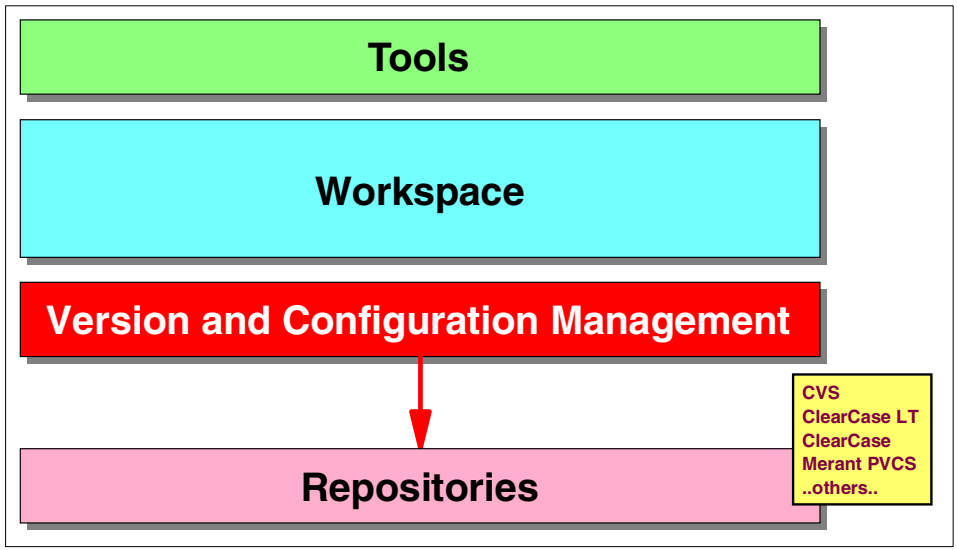


Figure 23-4 Version and configuration architecture of the workbench

23.2.6 Application Developer Team Support

Application Developer delivers adapters for two SCM products, **Rational ClearCase LT** and **Concurrent Versions System** (CVS). When you install Application Developer, you will be prompted to select which version control product you want to use. If you want to work in a team development environment, you must select either CVS or ClearCase LT, (unless you have installed, or plan to install, another third party SCM).

.

Attention: Neither of the SCM product servers will be installed automatically. The choice you make during the installation of Application Developer will only cause the corresponding client to be installed on your workstation. You have to install the corresponding server manually, either before you begin your Application Developer installation or after it has been completed.

The diagram in Figure 23-5 shows a comparison of the functionality of the two SCM products for which adapters are shipped with Application Developer.

SCM adapters for other commercial SCM products will have to be provided by the vendors of these products. To find a list of SCM products and adapters provided by IBM Business Partners, go to the SCM Adapters page of the product site (`http://www.ibm.com/software/ad/studioappdev/partners/scm.html`).

You can link from this list to the SCM product vendor's Web site to download the SCM adapter of your choice. IBM does not endorse or warrant these adapters. For support, or further information, please contact the SCM product vendor directly.

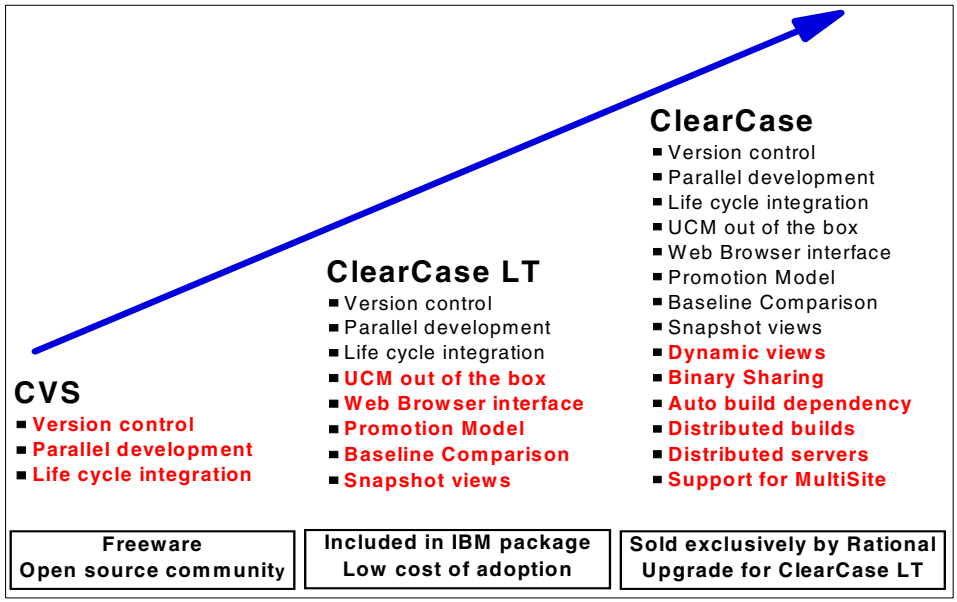


Figure 23-5 SCM tools functionality comparison

23.2.7 Terminology comparison

When working with Application Developer you will use the terminology provided by the team development environment. When you use facilities of versioning systems outside of Application Developer you will also need to understand their terminology.

Table 23-1 shows the mapping between terms used in Application Developer, CVS and ClearCase.

Table 23-1 Application Developer/CVS/ClearCase terminology comparison

Application Developer	CVS	ClearCase
Workspace	File System	Work area
Repository	Repository	VOB
Stream	Branch (tag)	Stream and project
Project	Folder	View

Application Developer	CVS	ClearCase
Resource	File	Element
Release	Revision	Check-in
Catch-up	Update	Compare with
Version	Commit (tag)	Version

The following chapters will discuss in more detail how to work with CVS and ClearCase LT.

24

Using CVS

Concurrent Versions System (CVS) is an open-source, network-transparent version control system, that can be used by individual developers as well as by large, distributed teams.

This chapter describes the following:

- What is CVS?
- Installing CVS
- CVS projects
- Team and resource perspectives
- Configuring a CVS repository
- Local and remote repositories
- Working with streams
- Team specific actions

© Copyright IBM Corp. 2002

24.1 How CVS works

One Software Configuration Management systems (SCMs) that can be used with Application Developer is Concurrent Versions System (CVS). This is an open-source, network-transparent version control system, that can be used by individual developers as well as by large, distributed teams. Some of the main features of CVS are:

- A *client-server access* method that lets developers access the latest code from anywhere that there is an Internet connection.
- An *unreserved check-out* approach to version control that helps to avoid artificial conflicts common when using an exclusive check-out model.
- C*lient tools* that are available on most common platforms.

CVS maintains a history of source code in a tree structure that shows all changes. Each change is stamped with the time it was made and the user name of the person who made it. As a rule, developers will also provide a description of the change. Given that information, CVS can help developers find answers to questions such as:

- Who made the change?
- When was it made?
- Why was it made?
- What other changes were made at the same time?

CVS records all changes to a given project in a directory tree called a *repository*. CVS doesn't work with ordinary directory trees; you need to work within a directory that CVS created for you. Just as you check out a book from a library before taking it home to read it, you use the `cvs checkout` command to get a directory tree from CVS before working on it.

Once CVS has created a working directory tree, you can edit, compile and test the files it contains in the usual way. From the perspective of the developer they are just files.

Since each developer uses a personal working directory, the changes that are made to it aren't automatically visible to the other developers in the team. CVS doesn't publish any changes until the developer has indicated that they are ready. When the changes have been tested, they must be explicitly committed to the repository to make them available to the rest of the development team.

What happens if more then one developer has changed the same file or the same line? Whose changes should prevail? It is generally impossible to answer this question automatically and CVS doesn’t make any attempts to resolve such a conflict.

It's important to realize what CVS does and doesn't consider a conflict. CVS's understanding of conflicts is strictly textual and it does not understand the semantics of your program. From its perspective your source code is simply *a tree of text files*.

Many version control systems lets a developer *lock* a file to prevent other developers from making changes to it until the lock is released. While locking may be appropriate in some situations, it is not as a general rule a “better” solution than the approach taken by CVS. Changes can usually be merged correctly, and developers do occasionally forget to release locks causing unnecessary delays.

Furthermore, locks prevent only textual conflicts; they do not prevent semantic conflicts if the two developers make their changes to different files.

The CVS server code is available at `http://www.cvshome.org`. Please refer to this site for information about how to install and configure your CVS repository, including user access and passwords. (24.3.1, “Installing CVS” on page 502 briefly describes the steps to install CVS.)

Application Developer currently supports two authentication protocols when using CVS: `pserver` (password server protocol) and `ssh`. The default is `pserver`.

The *CVS Manual* has been integrated into the Application Developer online help system and can be accessed from there.

24.2 SCM perspectives

Application Developer provides two perspectives where you can perform the relevant team and resource management tasks. These are the Team perspective and the Resource perspective.

These two perspectives are described in “Team perspective” on page 100 and “Resource perspective” on page 37. Figure 24-1 and Figure 24-2 show the default layout of the perspectives.

24.2.1 Team perspective

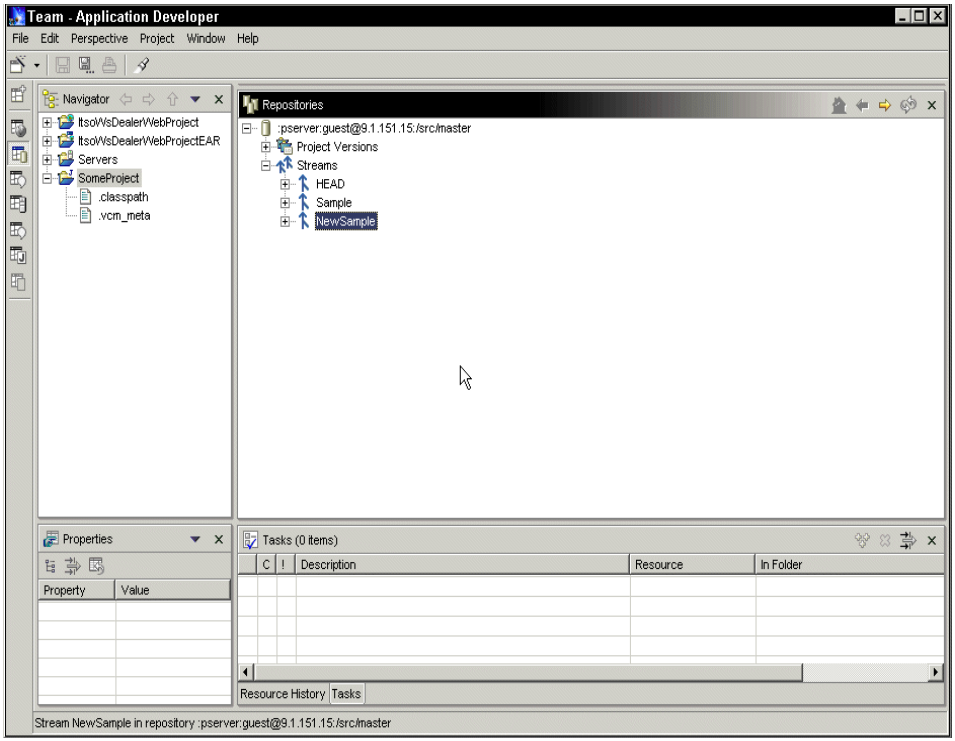


Figure 24-1 Team perspective

The Team perspective contains four panes:

- **Top left**: Shows Navigator view.
- **Top right**: Shows Repositories and Synchronize views.
- **Bottom left**: Shows Properties view.
- **Bottom right**: Shows Tasks view and Resource History view.

The Repositories view displays the repository connections, the project versions, and the active project streams with the projects.

The Synchronize view displays the changes between files in the local workspace and the team stream.

The Resource History view shows the sequence of changes performed on a file.

24.2.2 Resource Perspective

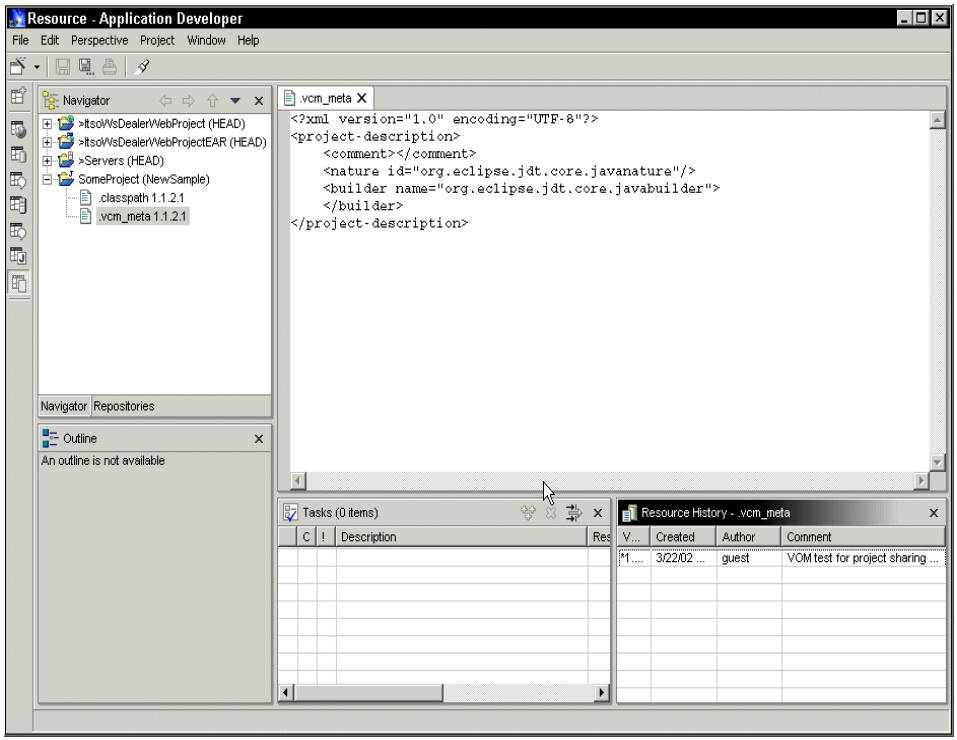


Figure 24-2 Resource perspective

The Resource perspective contains four panes:

- **Top left**: Shows Navigator and Repositories views.
- **Top right**: Reserved for editors of the selected resource.
- **Bottom left**: Shows Outline view of the resource opened in the active editor.
- **Bottom right**: Shows Tasks view and Resource History view.

24.3 Creating a CVS repository

We will now look at how to install CVS and how to create and use a repository.

24.3.1 Installing CVS

The first task is to install CVS. There are two ways to run CVS on a Windows platform.

1. As a client talking to a CVS server on a Unix/Linux box. This is the recommended setup and is the one most commonly used.
2. As a "local" or "non-client/server" CVS. This lets you run CVS if you have only Windows machines.

Due to some issues with local CVS on Windows, and with the suitability of Windows as a server operating system in general, CVS providers generally recommend using the second option only if you are evaluating CVS to get to know its functionality and not for production use.

Setting up an environment with CVS on Windows platform is quite straightforward:

1. Get the freely available code from the CVS Web site:

   `http://www.cvshome.org` — for source and binaries
   `http://www.cvsnt.com` — for Windows binaries for NT service

2. The Windows version of CVS comes with an installer. Double-click the installer and install it. We used version 1.11.1.3 as follows.

   `CVSNT_1.11.1.3.exe`

3. You need to add a path to the `CVSNT.EXE` (Figure 24-3):

   Control Panel-> System-> Advanced-> Environment Variables-> System Variables -> path

   For example:

   ```
   c:\Program Files\CVS for NT;
   ```

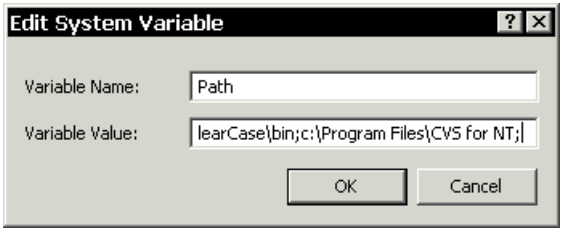


Figure 24-3 Adding a path

4. Open **CVSNT applet** from the control panel 

and create the repository.
   a. Go to the Repositories page (see Figure 24-4).
   b. Click Add
   c. Enter or select your repository directory

d. Initialize your repository, if you did not do this already.
e. Then apply.

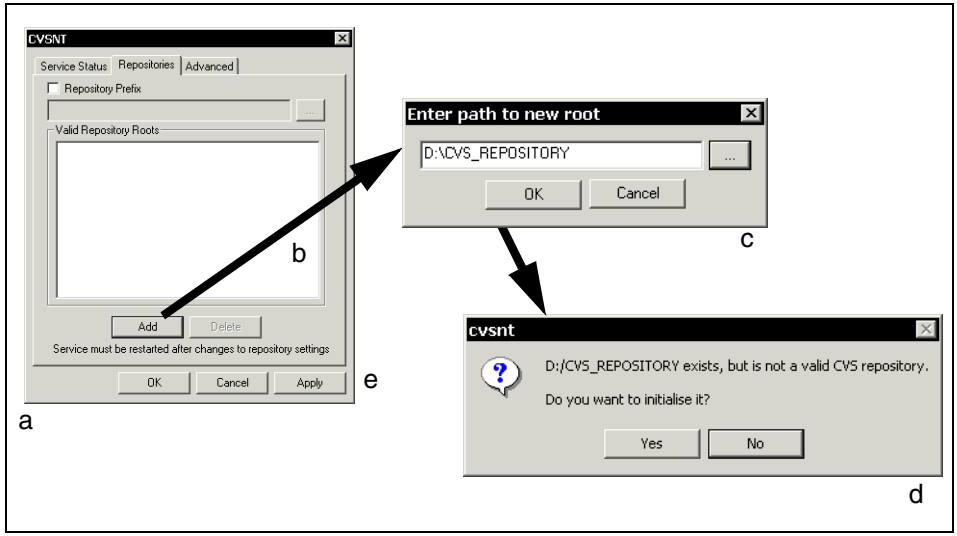


Figure 24-4 Creating a repository

5. Check the *Advanced* setting if you need. CVS will use port 2401. You can change it from the advanced setting.
6. Go to the *Service Status* page and click **Start** to start the service (Figure 24-5).

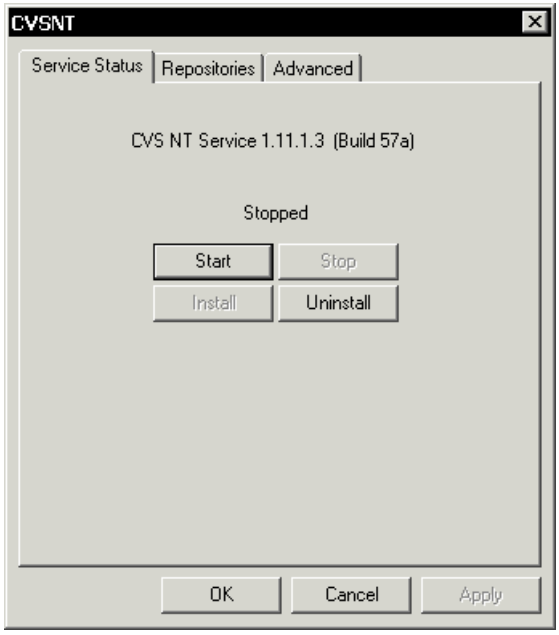


Figure 24-5 Start the CVSNT Service

7. If your CVS for NT does not come with an installer, follow the steps below:
   a. Unzip the code into a product directory, for example:

      ```
      d:\CVSNT
      ```

   b. Create the **repository directory** and initialize the repository:

      ```
      cd d:\CVSNT
      cvs -d :local:x:/CVSRepo init
      ```

      (Where, x stands for your target drive)

   c. Create a **Windows service** that can be started and stopped:

      ```
      ntservice -i x:/CVSRepo
      ```

      (Where, x stands for your target drive)

   d. Start the **service**.

Note: If you are going to install CVS server on Unix as a service, you need to set up your service. For example, the repository is located on /src/master,

Modify the inetd.conf to run the CVS as the service.

```
cvspserver stream tcp nowait root /usr/local/bin/cvs cvs
--allow-root=/src/master pserver
```

Modify the service and assign the port 2401 to the CVS service.

```
cvspserver       2401/tcp                       # CVS pserver
```

24.3.2 Connecting to a repository

Before you can share your project with other users you must first specify which one of the available repositories you want to work with. There are two ways that you can do this in Application Developer.

The first one is to do so using the context menu from the Repositories view (in the Resource or Team perspective).

In the Resource perspective you can switch to the Repositories view (in the top left pane). In the Team perspective you see the Repositories view in the top right pane.

When you are connecting to a repository for the first time, the Repositories view will be empty.

Display the context menu for the view and select **New—>CVS Repository Location** (Figure 24-6 and Figure 24-7).

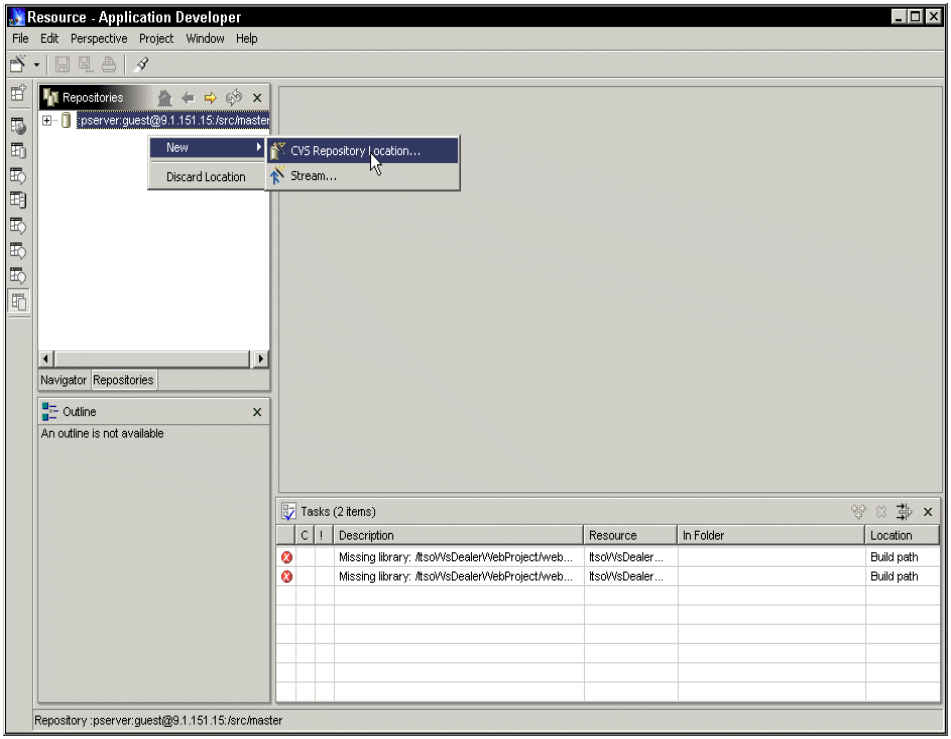


Figure 24-6 Connecting to a new repository from Resource perspective

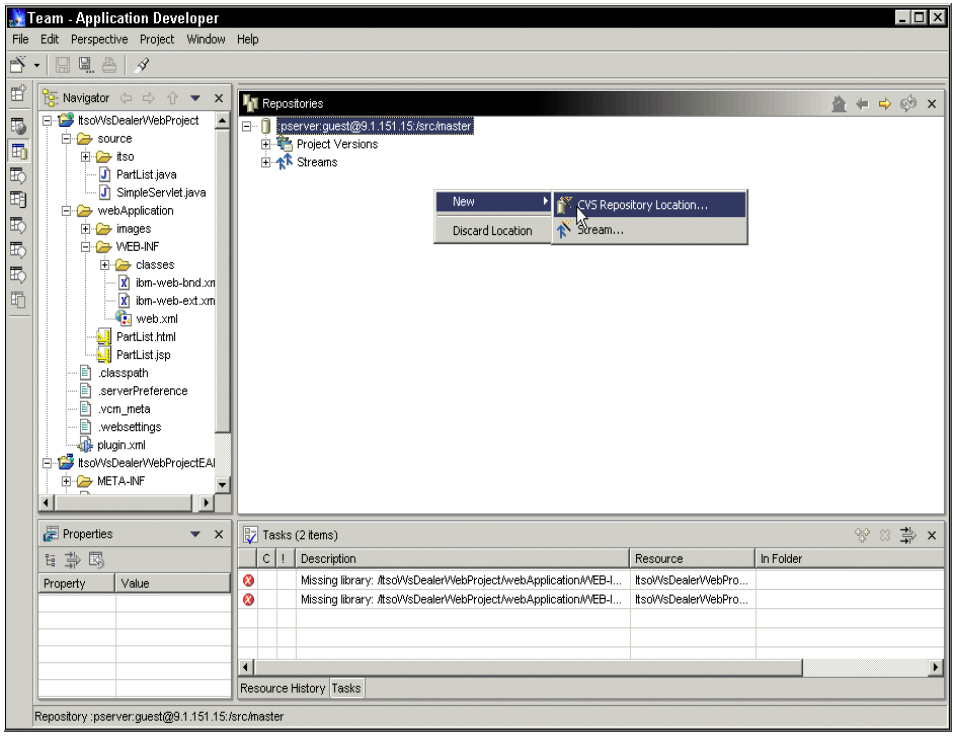


Figure 24-7 Connecting to CVS repository from the Team perspective

The second possibility is to click the **Open The New Wizard** icon  , select **Other**, and then select **CVS** in the left pane and **Repository Location** in the right pane (Figure 24-8).

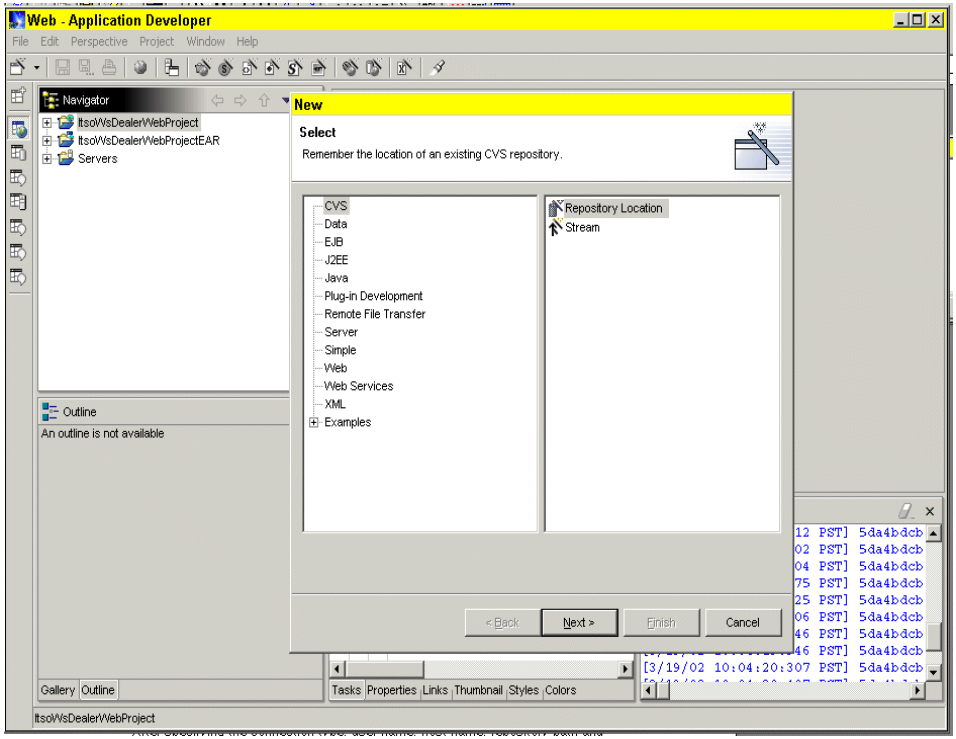


Figure 24-8 Connecting to new CVS repository using Open The New Wizard

In either case the CVS Repository Location wizard will be opened (Figure 24-9).

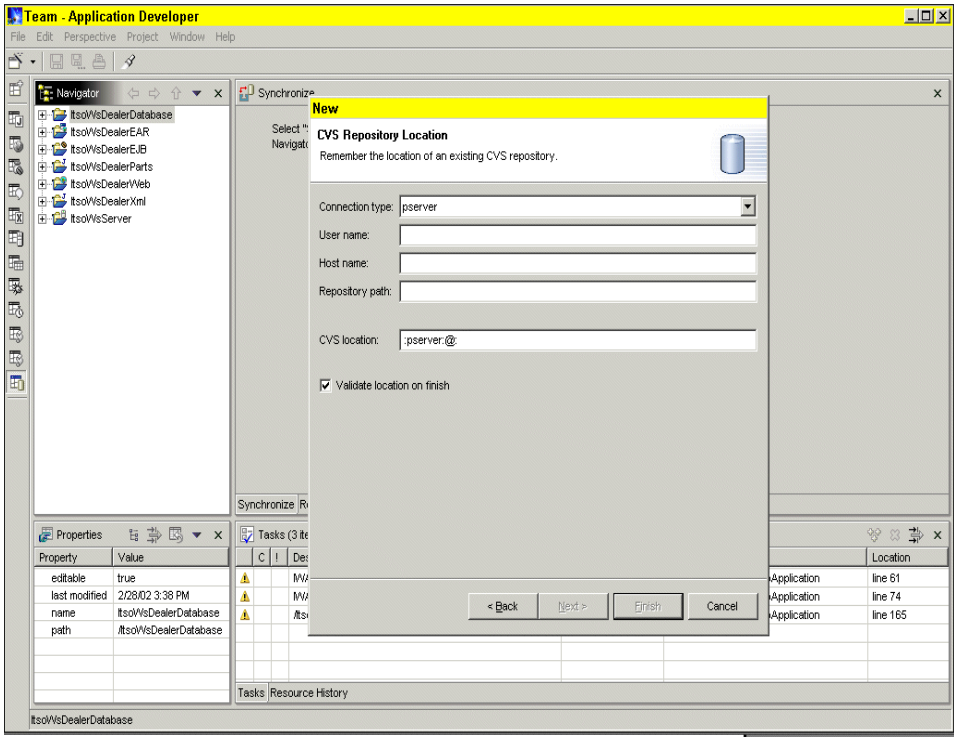


Figure 24-9 CVS Repository Location wizard

In the CVS Repository Location wizard you need to fill in the location of the repository and your login information:

> **Note:** When using a client/server configuration you may require assistance from your repository administrator in order to provide the required information.

- In the **Connection type** field, select the type of CVS connection for the repository. Choose `pserver`.
- In the **User name** field, type the user name with which you want to connect.
- In the **Host name** field, type the address of the host (for example `hostmachine.com`)
- In the **Repository path** field, type the path for the repository at the host address (for example `D:/cvsroot/repositoryName`).

The CVS location field shows the canonical CVS location string. This string is often found in CVS documentation to denote how to connect to a repository. This field automatically updates with the connection information from the other fields in the dialog. You do not need to specify additional information in this field. Optionally, if you already have a CVS location string, you can paste it in here instead of editing the individual fields.

- By default the **Validate location on finish** check box is checked.
- Click **Finish** when you are done.

Since you have checked **Validate location on finish**, the wizard will now attempt to validate the information by connecting to the repository. In doing so it may prompt you for your password. Note that this repository connection will only be used to validate the information.

The Repositories view should now show the new repository location (Figure 24-10).

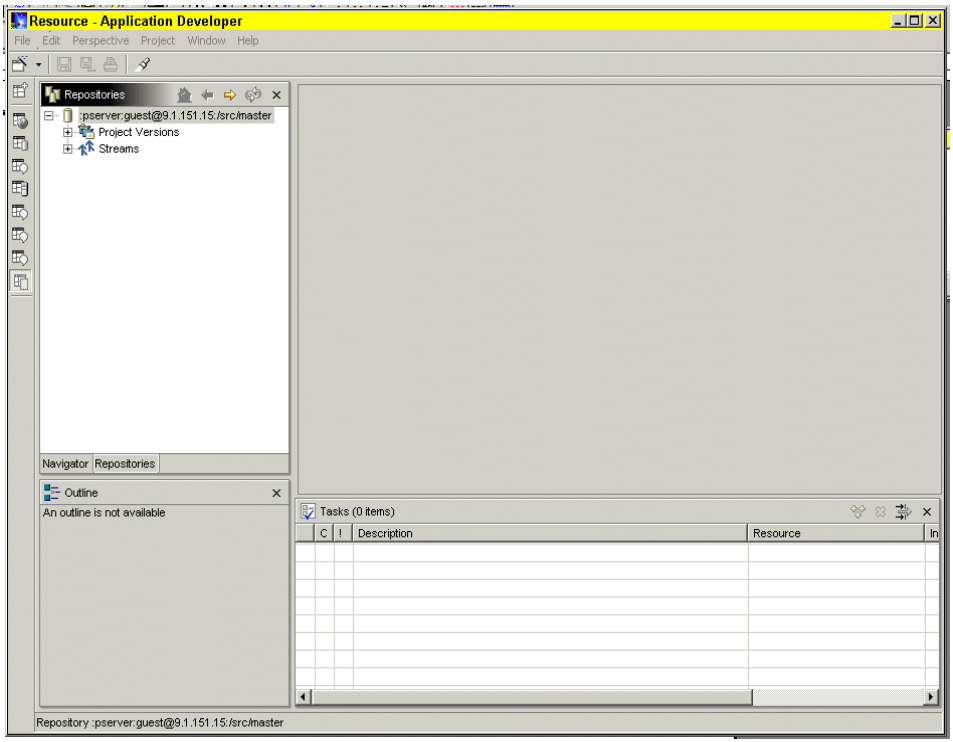


Figure 24-10 Repositories view in Resource perspective

This view shows repository locations added by the user. Expanding a location will show the Project Versions and Streams in that repository.

You can expand the Project Version and Stream nodes to show the folders and files contained within them. From this view you can also add resources to the workbench, open editors on files in the repository, view history, and compare resource versions.

24.4 Local versus remote repository

The CVS repository stores a complete copy of all the files and directories that are under its version control.

Normally you would never access any of the files in the repository directly. Instead, you use CVS commands to get your own copy of the files into a *working directory* and then work on that copy. When you've finished a set of changes, you check (or *commit*) them back into the repository. The repository then contains the changes which you have made, as well as recording exactly what you changed, when you changed it, and other such meta information.

Note that the repository is not a subdirectory of the working directory or vice versa. They should be in separate locations.

CVS can access a repository by a variety of means. It might be on the local computer or it might be on a computer across the room or across the world. Using CVS in the second way is known as *client/server* operation. In this case you run CVS on a machine which can mount your working directory, known as the *client*, and tell it to communicate with a machine which can mount the repository, known as the *server*. Generally, using a remote repository is just like using a local one, except that the format of the repository name is different.

To distinguish the different ways to access a repository, the repository name can start with the *access method*.

For example, the access method `:local:` means to access a local repository directory, so the repository `:local:/usr/local/cvsroot` means that the repository is in /usr/local/cvsroot on the computer running CVS.

If the access method is omitted, that is if the repository does not contain '`:`', then `:local:` is assumed. If it contains only '`:`' then either `:ext:` or `:server:` is assumed.

For example, if you have a local repository in /usr/local/cvsroot directory you can use `/usr/local/cvsroot` instead of `:local:/usr/local/cvsroot`. But if (under Windows NT, for example) your local repository is c:\src\cvsroot, then you must specify the access method, as in `:local:c:\src\cvsroot`.

The repository directory itself is split into two parts. '$CVSROOT/CVSROOT' contains administrative files for CVS. The other directories contain the actual user-defined files.

24.5 Streams in CVS

This section explains how Application Developer streams map to a CVS repository. There are a few particularities that need to be explained:

- Streams in Application Developer map to *branches* in CVS. The CVS repository always has a *default branch* called HEAD. This is the main trunk in the repository. Typically development is done in the HEAD stream.
- There is no easy way to query a CVS repository for all existing streams. This means that when a repository connection is made to a CVS repository, only HEAD will be shown

  Creation of a new stream in the repository view can be viewed as defining a new stream name.

  Note: The underlying CVS branch is not created until you explicitly copy a project version or release contents to the stream. This is a particularity of CVS.

  Stream name definitions are persisted between workspace sessions until explicitly removed within the repository view.
- Advanced CVS users can use the Resource History view to see existing branch tags. All CVS tags are displayed in brackets to the right of the revision number (for example, 1.2 (v1, merge1, v2)).

  Since streams are created as branch tags in CVS you can view the existing branch tags from the file history and then create a stream by that name to access an existing branch.
- There is no easy way in CVS to determine if a project is part of branch, other than examining all files in the project for the appropriate branch tag.

  Consequently, when you create a new stream, the names of all existing projects in the repository will show up as children of the stream node in the repository browser tree.
- Removing a stream from the repository browser can be seen as removing the definition from the workbench. The underlying CVS branch is left untouched.

Since streams represent a basic concept in CVS, their usage patterns (for example when and why to use them) are clearly described in the CVS specific documentation regarding branches.

24.5.1 Creating a new stream

As we have already mentioned, development is typically done in the HEAD stream.

In some circumstances it may be necessary to work with other streams. For example you may require an additional stream for ongoing maintenance.

To add a stream to a repository, go to the Repositories view and select the repository where you want to create the new stream.

> If there are no repository locations listed in the Repositories view you need to create one see 24.3.2, “Connecting to a repository” on page 505.

From the context menu select **New—>Stream** (Figure 24-11).

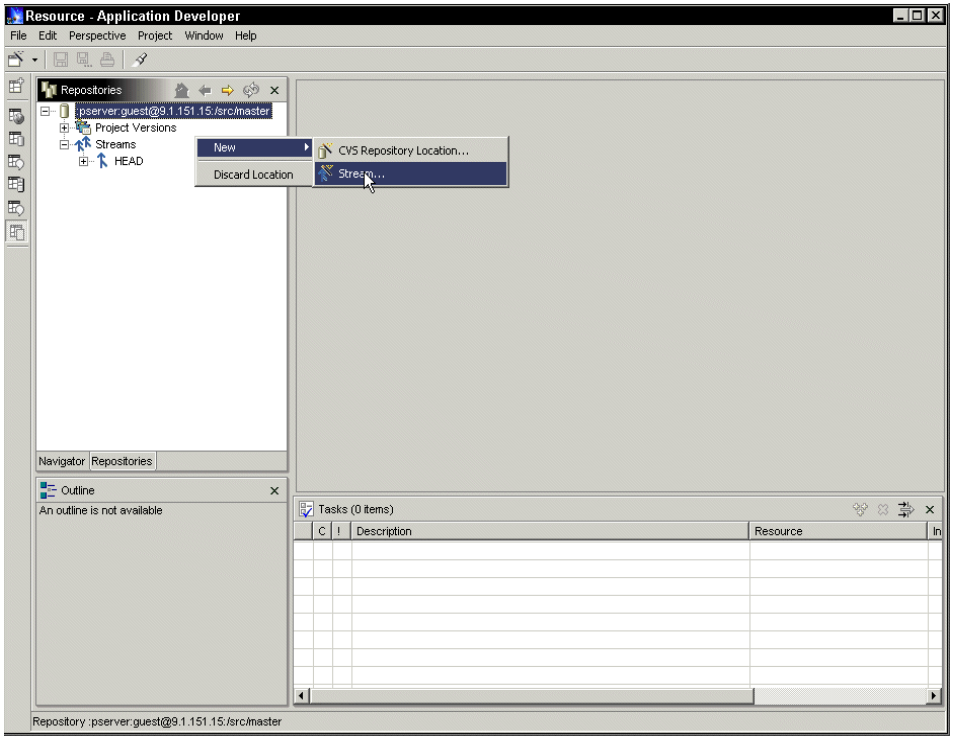


Figure 24-11 Adding a stream to repository

In the New Stream dialog enter the name of the stream you want to work with (Figure 24-12).

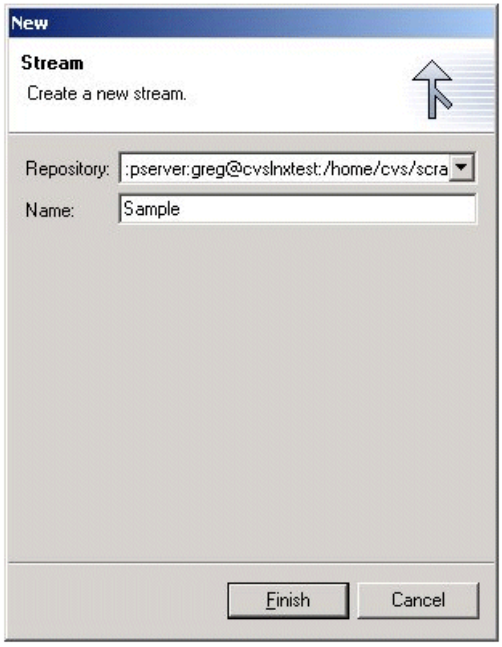


Figure 24-12 New stream dialog

> **Important:** This operation does not actually create a stream, rather it remembers the name of a stream you are interested in. The stream (CVS branch) will be created the first time you release resources to it.
>
> If other users want to work in the same special stream they will need to follow the exact same steps we have done here. Until they have actually instructed the workbench to add a new stream name, the stream will not appear under Streams in their Repositories views

As was mentioned earlier, CVS repositories do not provide an "all used stream names" view. Consequently you must explicitly tell the workbench the name of each special stream you want to work with.

8. In the Repositories view expand Streams and observe that it now contains the new Stream "Sample". In the future when releasing your projects to the repository, you will now have two streams to choose from, HEAD and Sample.

24.5.2 Viewing stream resource history

You can view the history of a stream resource in a repository in the Repositories view. There are a couple of prerequisites that need to be fulfilled before you can perform the following steps:

- You must have the given repository location defined in your workspace.
- You must be an authenticated user on the server.

1. In the Repositories view, expand the repository location that contains the resource for which you want to view the history.
2. Select the resource.You can do this in either:
   - A project version in **Project Versions**.
   - A project version in a stream in **Streams**.
3. Select **Show in Resource History** from the context menu (Figure 24-13).

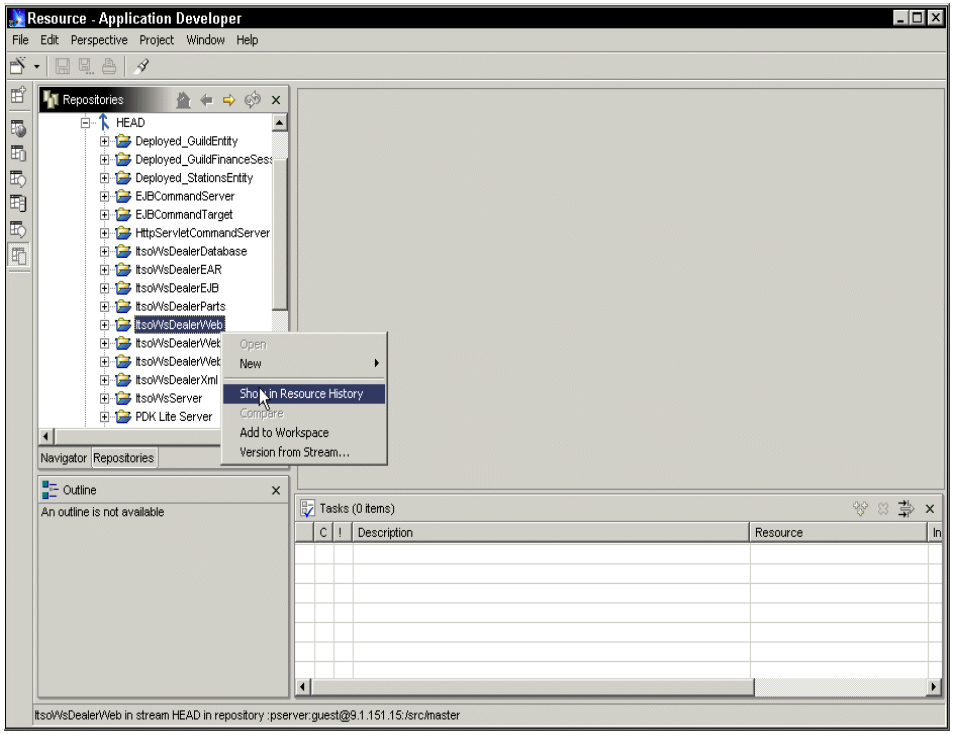


Figure 24-13 Triggering the Show Resource History menu item

4. In the Resource perspective a new Resource History view will be added to the bottom right pane. Here you will see information about each version of the stream resource (Figure 24-14):

- **Version**: The version consists of a repository generated version number followed by a list of additional version information. In CVS these are the tags associated with that version.
- **Date Created**: When the version was created (released).
- **Author**: Who created (released) this version.
- **Version Comments**: If any were added by its author.

Note: The CVS server may not support history. In this case you will not see the history information specified above.

You might need to maximize this view in your perspective to be able to see the complete information.

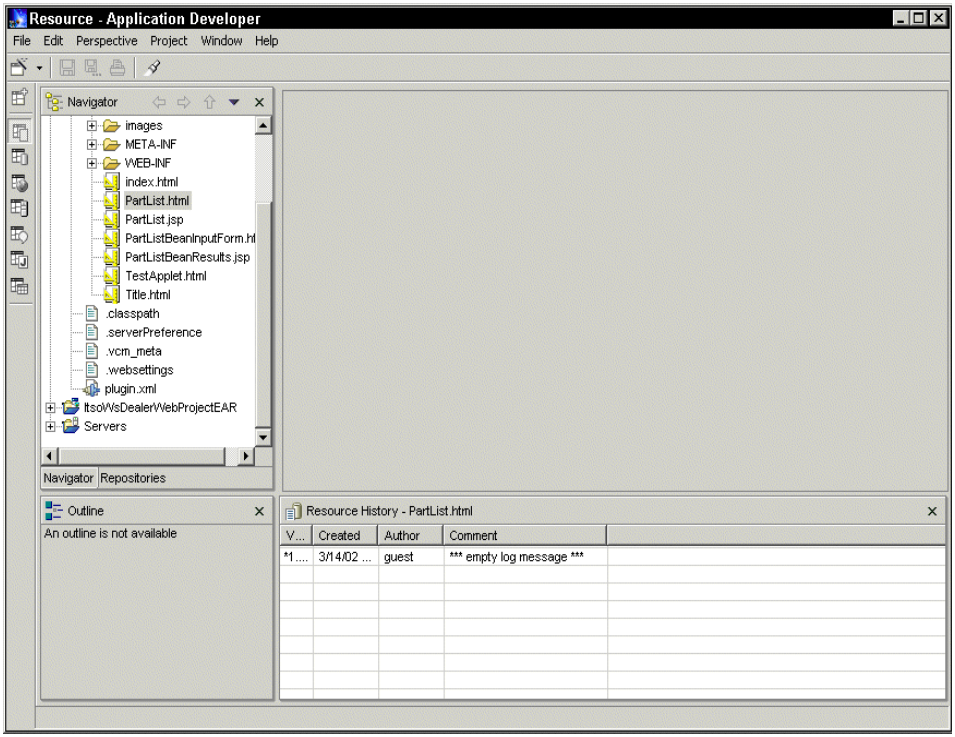


Figure 24-14 Viewing the resource history

24.5.3 Associating a project with a stream

To change the sharing of a project, you have to change the stream with which the project is associated.

There are two ways to change the sharing for a workspace project:

1. Adding the project from the stream to workspace via the Repositories view.
2. Changing the sharing of the project (the repository and stream through which it can be shared) via the Team page of the property dialog of the project.

Note: You need to be careful when changing the sharing of a project. Adding a project from the stream to the workspace via the Repositories view or changing its sharing to another stream in another repository, deletes all existing synchronization information currently managed for the given project and re-initializes it with the new information for the newly added/reassigned project.

Changing the sharing to another stream in the same repository does not change the synchronization information. Changing sharing to another stream in the same repository via the project's property page is used when you are splitting and releasing the changes into another stream.

To change the stream that a project is associated with you need to do the following:

1. Select the project in the Navigator view.
2. Select **Properties** from the context menu.
3. In the Properties dialog, select **Team** in the left pane and click the **Change** button (Figure 24-15).

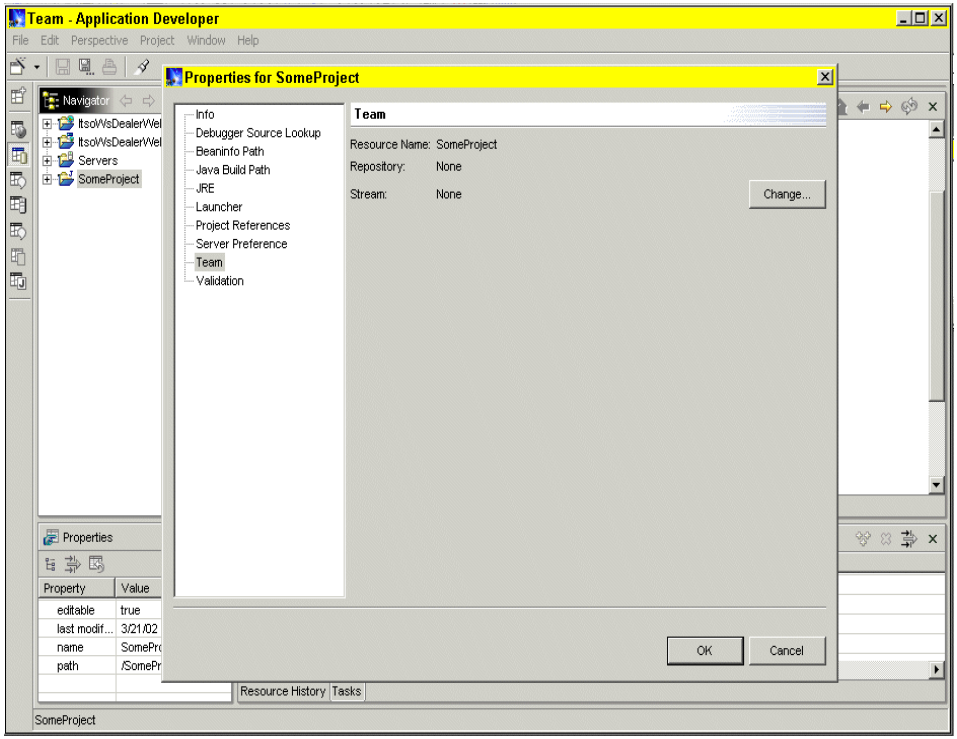


Figure 24-15 Team page from the project properties dialog

4. In the Set Project Sharing dialog select the repository and stream with which you would like the project to be associated, then click the **OK** button in the dialog (Figure 24-16).

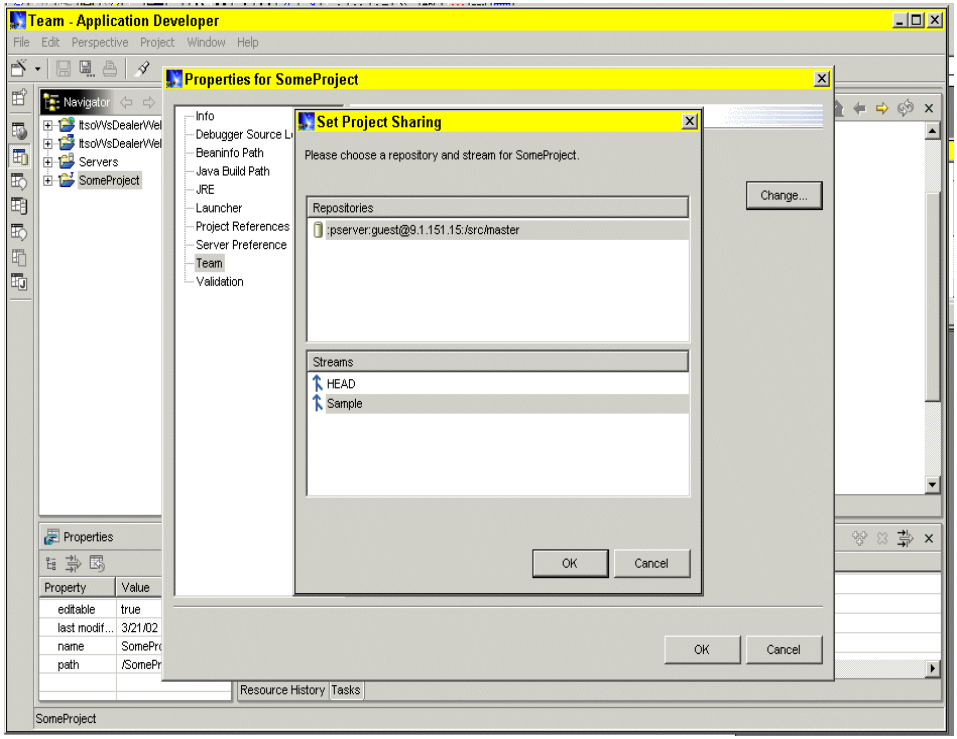


Figure 24-16 Assigning the project to another repository or stream

5. Click the **OK** button in the Properties dialog.

24.5.4 Splitting from a stream

Typically you will want to split a stream if you plan to do one of the following:

1. Fix a bug in an previous version of a project.
2. Store project variants or a developer's private work in a separate stream.

When splitting a stream you always have to split at a pre-defined point in time. This will become the *initial state* of the new stream.

Note: When you split a stream, the initial state will always be a version.

There are two options for splitting a stream:

1. **Splitting based on a project version**: Choose this option if you are splitting based on an existing project version. This could for example be done to fix a bug in a previous version of your project.

2. **Splitting with changes in your workspace**: Sometimes you don't want to release your changes to everyone in the team but would still like to version them. You may want to do this, for example, if your changes aren't stable yet, or if it will take a long time before you are ready to release them to the team. In this case you may want to create a new stream to release your changes to.

Splitting based on project version

To split a new stream from an existing project version do the following:

1. Bring up the context menu in the Repositories view and select **New—>Stream** to bring up the New Stream wizard.
2. In the New Stream wizard Figure 24-17, select the repository that should contain the new stream. Enter a name for the new stream and click **Finish**.

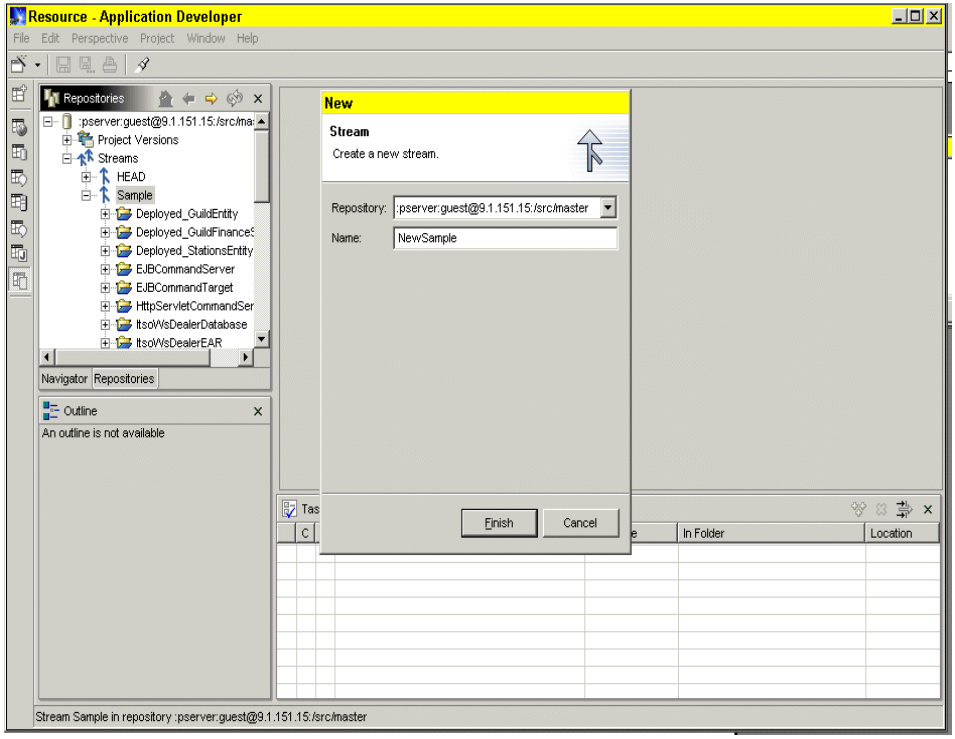


Figure 24-17 New stream wizard

The new stream will be added to the repository you have specified. (In our example we used the same repository so the new stream was added to it.)

3. Now we can add a project version to this new stream (this will set the initial state of the stream). In the Repositories view, select the new stream and select **Copy Version To Stream** from the context menu (Figure 24-18).

4. A Choose Version dialog will appear. Select the projects and versions that you want to copy to the new stream and click **OK** to confirm (Figure 24-18).

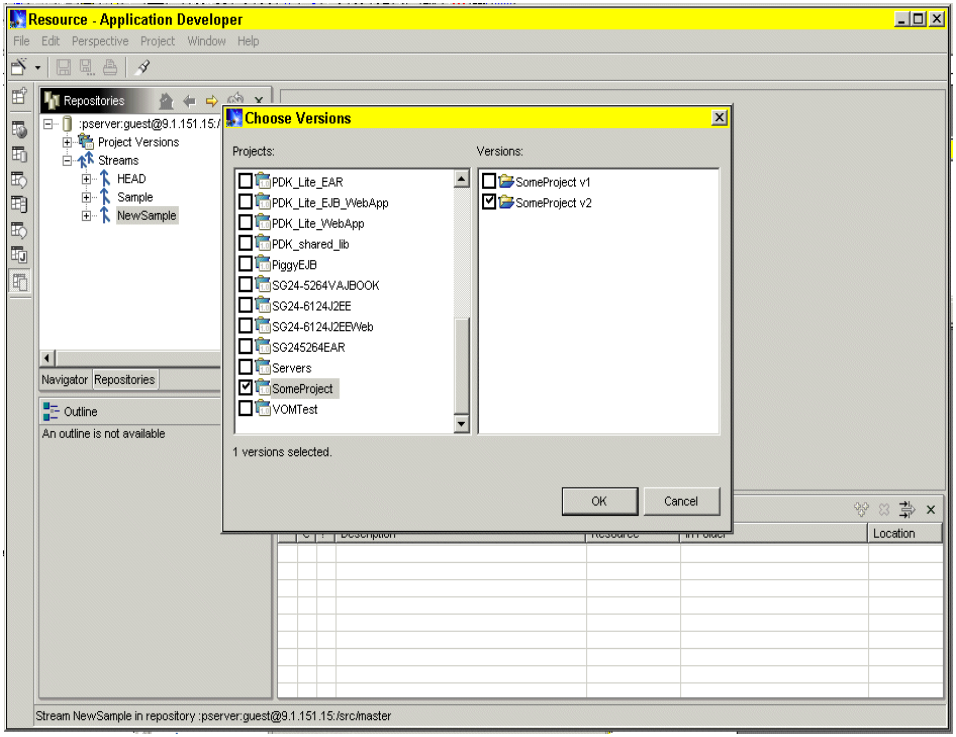


Figure 24-18 Selecting projects and versions to be copied to the new stream

5. You can now expand the new stream to see its projects. You will see the resources that you have just added to the new stream.
6. Select the project you want to add to your workspace and select **Add to Workspace** from the context menu.You need to do this to ensure that from now on all released files for this project will go to the new stream.

Tip: If you enable **Show Version Info** via the menu in the navigator's toolbar, you can see the new stream name shown to the right of the project (Figure 24-19).

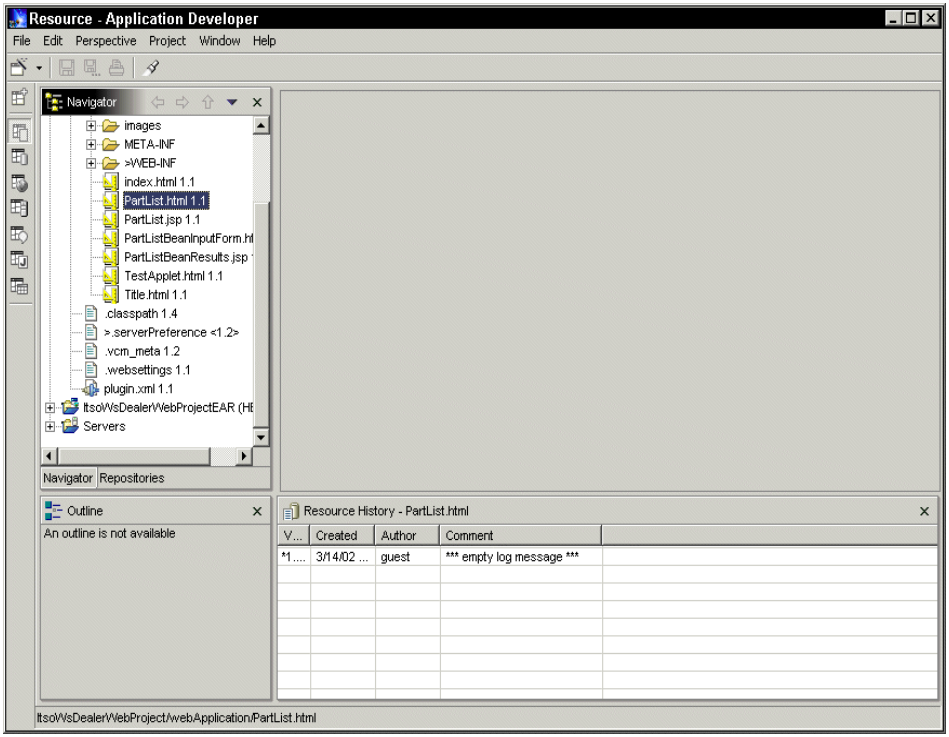


Figure 24-19 Navigator view with Show Version Info on

Splitting with changes in your workspace

To split a new stream containing the changes you have just added in your workspace you have to do the following:

1. Create a version of your project from the workspace to capture the state before you have made any changes. This version will contain all the base versions of your project's resources. The changes you have made in your workspace to these base versions will later be released to the new stream.

   To do so, select the project in the Navigator view and select **Version From Workspace** from its context menu.

2. Complete the Version Selected Resources dialog. While performing the versioning, a dialog will warn you of your outgoing changes as not being released. This is expected. Review the changes listed in the details window.

3. Create a new stream. Bring up the context menu for the Repositories view and select **New—>Stream** to bring up the New Stream wizard.

4. In the New Stream wizard, select the repository that will contain the stream you are going to split. Enter a name for the new stream and click **Finish** (Figure 24-17).
5. Add the project version to the new stream. (This sets the initial state of the stream.) In the Repositories view, select the new stream and select **Copy Version To Stream** from the context menu. A version selection dialog will be displayed (Figure 24-18). Select the project and the version you have created in Step 2.
6. Select the project in the Navigator view and change the sharing of the project to the new stream. See “Associating a project with a stream” on page 516 about how to do so.
7. Now you can synchronize your project with the new stream. Your changes will appear as outgoing to the new stream. The list should be identical to the list of changes not being released in Step 2. Release them now into the new stream.

This task is now completed.

24.5.5 Merging from a stream

Merging involves identifying changes that have been made between two points in a stream, the initial state and the end state, and merging them into your workspace.

Typically the initial state will be the root of a stream (project version) and the end state can either be the stream or another project version.

Note: It is important to realize that the destination of the merge is always the project in your workspace. After the merge has completed you should first test the changes locally and only then release them to the new stream.

To merge a project:

1. Add the project you want to merge from the destination stream into your workspace. To do so, select the project to merge in the Repositories view and select **Add To Workspace** from its context menu.
2. In the Navigator view, select the project and select **Team—>Merge** from its context menu. This will bring up the Merge wizard. This wizard helps you to merge changes between two states of a project into your workspace.
3. In the Merge Setup page, choose which project in the workspace you want to merge and the repository in which the project resides and the click **Next** (Figure 24-20).

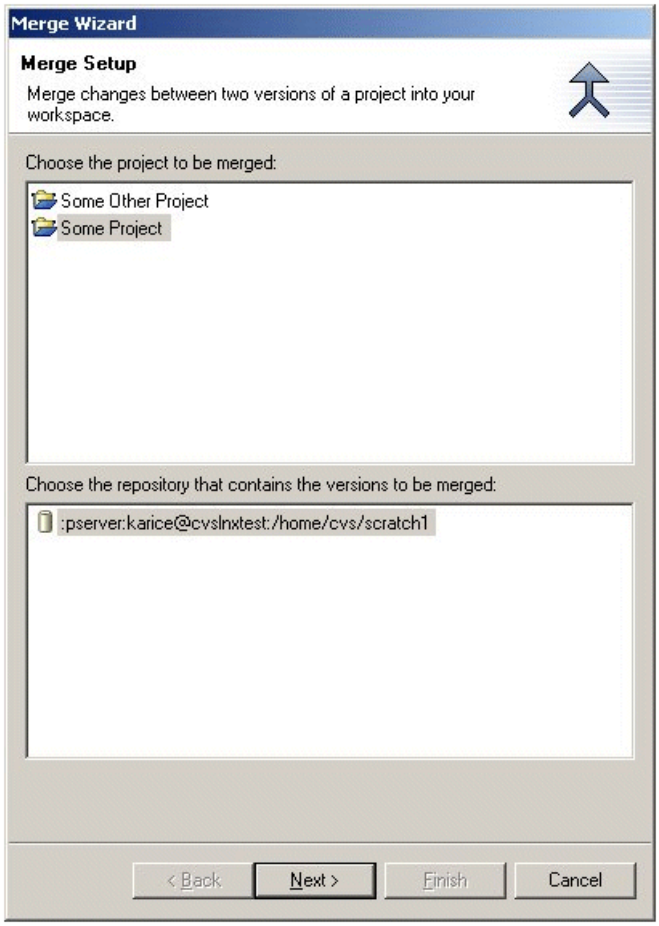


Figure 24-20 Setup page of the Merge wizard

4. On the Initial State page, choose the initial state of the merge. Usually this will be the root project version on which the originating stream was split. Select a version in the list and click **Next** (Figure 24-21).

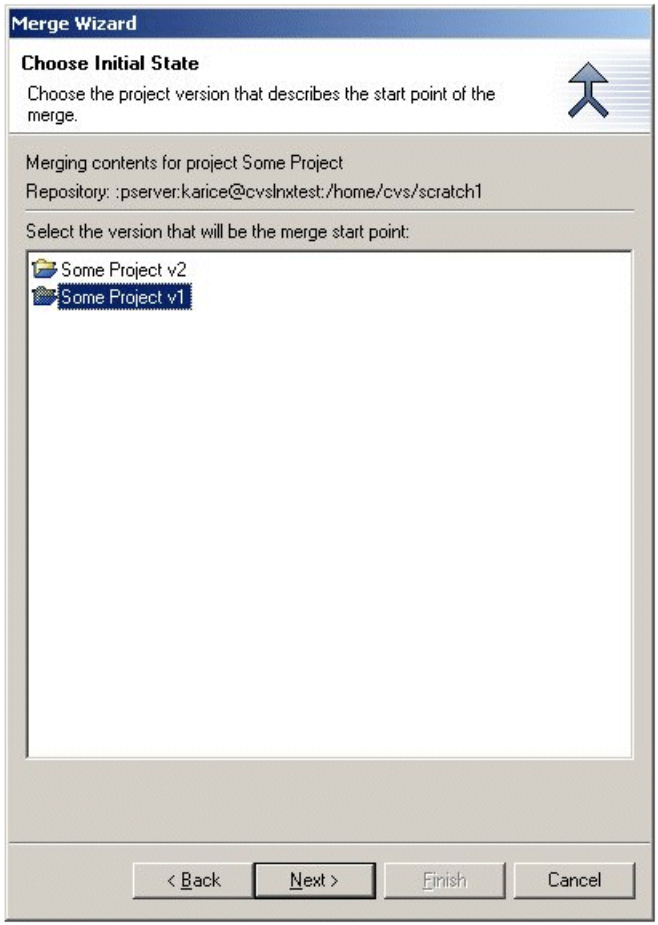

Merge Wizard
Choose Initial State
Choose the project version that describes the start point of the merge.
Merging contents for project Some Project
Repository: :pserver:karice@cvslnxtest:/home/cvs/scratch1
Select the version that will be the merge start point:
Some Project v2
Some Project v1
< Back
Next >
Finish
Cancel

Figure 24-21 Initial State page of the Merge wizard

5. On the End State page, choose the end state of the merge. This can be either a project version (for example if you versioned from your stream before the merge) or it can be a stream. Click **Finish** (Figure 24-22).

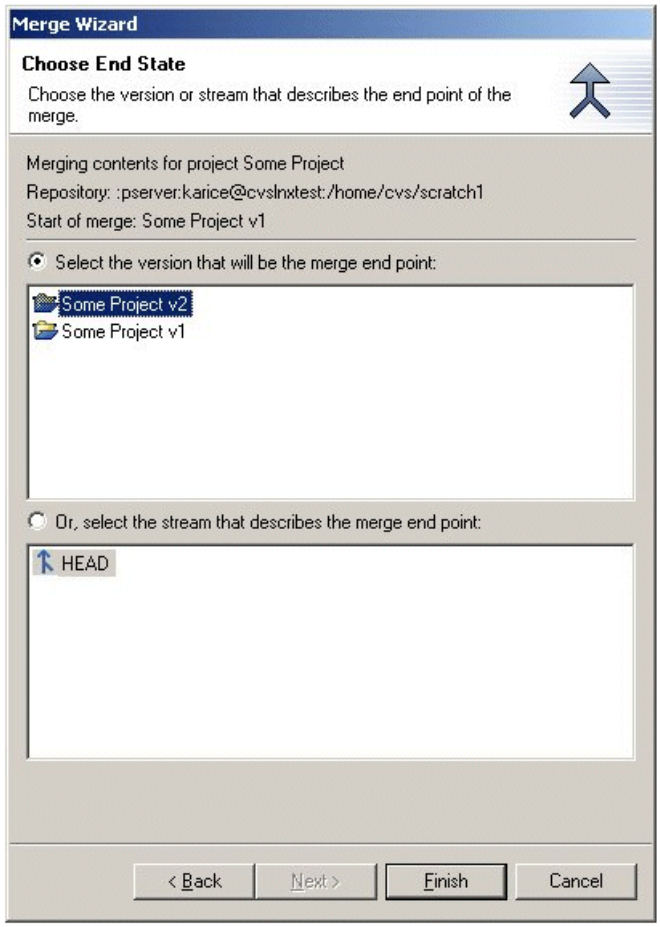

Merge Wizard
Choose End State
Choose the version or stream that describes the end point of the merge.
Merging contents for project Some Project
Repository: :pserver:karice@cvslnxtest:/home/cvs/scratch1
Start of merge: Some Project v1
Select the version that will be the merge end point:
Some Project v2
Some Project v1
Or, select the stream that describes the merge end point:
HEAD
< Back
Next >
Finish
Cancel

Figure 24-22 End State page of the Merge wizard

6. A merge view will appear in the editor area showing the changes between the initial state and the end state based on the resources in your workspace. Catch up or merge changes within this view to your local workspace (Figure 24-23).

Note: You cannot release changes from within the merge view.

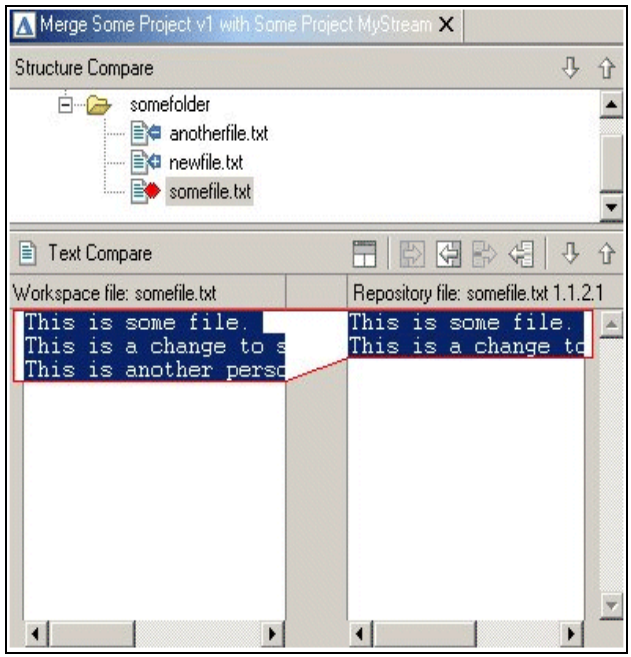


Figure 24-23 The merge view

7. After merging the changes, close the Merge view and test and review the result of the merge.
8. After merging changes into a stream, it is recommended that you create a project version to save the state of the project after the merge. This project version can be used to split again or be used with subsequent merges.

This task is now completed.

24.6 Team specific actions

It is not easy to adequately demonstrate on paper how to work in a multiuser development. This is because you need at least two concurrent developers using the same shared repository. To be able to follow the exact steps of the process, you would have to do the same, executing the examples step by step.

For that reason the examples that we will be using do not represent a continuous process, but rather they should be seen as samples of tasks that may be performed using the Application Developer team support. Taken together they will hopefully give you enough information to help you productively and effectively use the multiuser development environment.

In the following sections we will describe how to perform these tasks:

- Comparing files
- Catching up your project
- Versioning your project
- Releasing your project

24.7 Comparing the resources

You can compare two or three selected resources to view the differences between the resources. You can compare a file with one or two other files, a folder with one or two other folders, and a project with one or two other projects.

You can also compare one selected resource with other states in the local edit history or with other versions of that resource.

When a comparison is performed, all applicable compare editors appear in the editor area. Here you can browse through all the differences and copy/move the highlighted differences between the compared resources. You can then save changes to resources that were made in the compare editor.

You can compare resources in the workbench in several different ways:

1. **Compare two versions of the same resource in the local history**: When you select a resource in the workbench you can compare it to a version that is in the local history. The local history is a representation of each “edit and save” that has been made to the resource since it was added to the workbench.
2. **Compare a workbench resource with a version**: When you select a resource in the workbench, you can compare it with a version of the resource that has been released to the repository. This requires you to choose a version of the resource from a list.
3. **Compare a workbench resource with the current stream contents**: When you select a resource in the workbench, you can compare it with the version of that resource that is currently released to the stream.
4. **Compare a workbench resource with its base version**: When you select a resource in the workbench that has been modified, you can compare it with its state prior to modification. This is because the base version denotes the version you have loaded.

24.7.1 Three way compare

Three versions of a resource can be compared at once.

Three-way compare shows the differences between three different resources in the workbench or three versions of one resource. This feature is most useful when merging resources or when there a conflict is detected during synchronization.

When such a situation occurs, you will be able to view the differences between these three different resource versions:

1. The resource in the workbench
2. The version of the resource that is released in the stream
3. The common ancestor on which the two conflicting versions are based

The differences that will be highlighted show you what has been changed in the workbench resource as compared to the common ancestor, and what has been changed in the stream resource as compared to the common ancestor.

24.7.2 Comparing resources

To compare two or three resources:

1. Ctrl-click to multi-select two or three resources in the Navigator view.
2. From the resources context menu, select **Compare With—>Each Other**.
3. The Compare editor displays, showing the differences between the resources.

Understanding the comparison

Comparing two files (file1.txt and file2.txt in the following example) results in a compare editor as shown in Figure 24-24.

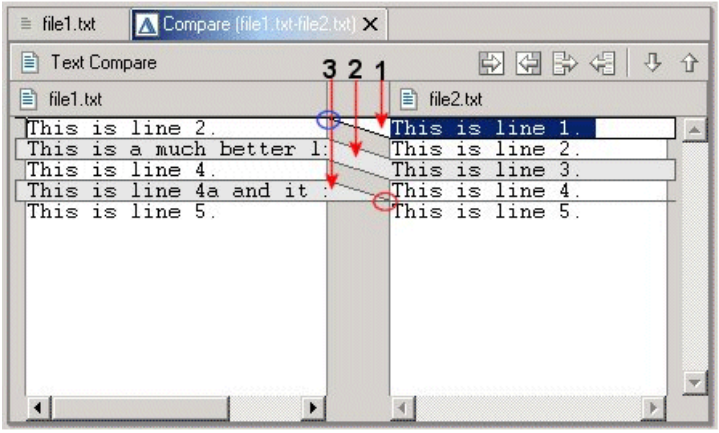


Figure 24-24 File (text) compare editor

The left side shows the contents of file1.txt and the right side shows the contents of file2.txt. The lines connecting the left and right panes indicate the differences between the files.

You can double-click on the editor tab to maximize the editor.

Comparing with repository versions

To compare workbench resources with versions in the repository you have to:

1. Select a resource in the Navigator view.
2. Bring up the context menu. Now you can select one of the following menu items:
   - **Compare With—>Stream Contents**: Compares workbench resource with the version currently released in the stream.
   - **Compare With—>Version**: Compares workbench resource with a version in the repository.
   - **Compare With—>Base Version**: Compares workbench resource with the version that was taken from the stream during the last synchronization.
3. In the compare dialog that will be opened, you can now browse to select the version with which you want to compare the workbench resource. Notice you can see the differences between the version and the workbench resource in the text compare area (Figure 24-25).
4. Click the **Done** button to close the compare dialog when you are finished.

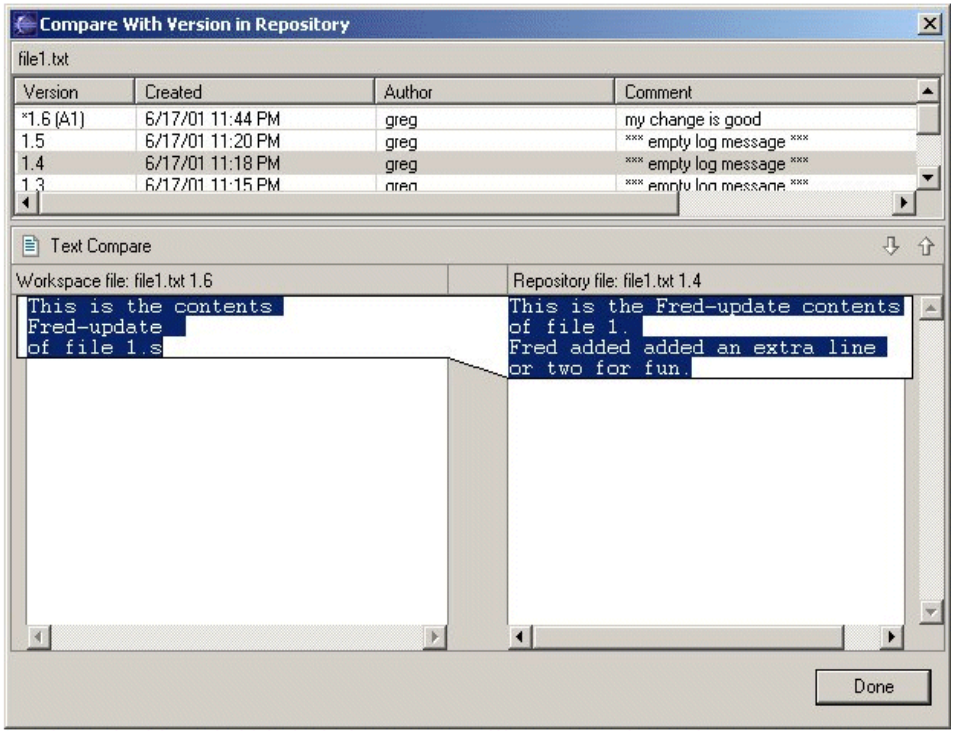


Figure 24-25 Version compare editor

Working with the comparison

There are two parts in the compare editor's local toolbar.

The right group of local toolbar buttons allows you to move to the next or previous change:

1. Click the **Select Next Change** button to select the next change.
2. Click the **Select Previous Change** button to move to the previous change.

The left group of local toolbar buttons allows you to merge changes from the left file to the right file and vice versa. There are four types of merges you can perform:

- Copy **whole document** from left to **right**.
- Copy **whole document** from right to **left**.
- Copy **current change** from left to **right**.
- Copy **current change** from right to **left**.

Typically the **Copy whole document** actions are used when you are sure that the entire file on either the left or right can just be replaced by the contents of the other file. The **Copy current change** buttons allow you to merge the single changes one by one.

24.8 Saving your work

Resources need to be versioned from time to time in order to capture a snapshot of their current state at a specific point in time. When a resource is versioned a *non-modifiable* copy of it is stored in a repository.

There are some important but subtle issues related to working with a repository.

- Each project is associated with a specific stream in a specific repository. Different projects can be associated with different repositories, which may in fact be on completely different servers.
- The stream contains all projects in the repository. Individual users can pick which projects they are interested in and add them to their workspaces. From that point on they are synchronizing those projects only with respect to the stream.
- The stream represents a large in-progress collection of all known projects. From the stream's perspective everything is always open for change.
- The act of versioning a project effectively snapshots it and places it into the Project Versions section of the repository, however the “stream copy” of it is still open for changes.
- When you version the project you should do so by versioning the project as it appears in your workbench. This is where you can easy control what it looks like and test it to ensure that there are no errors.

> **Note:** For this reason it is important to synchronize the project with the stream prior to versioning it.

If you don’t synchronize before versioning you might later discover that other users have released valid changes to the project that you have yet to catch up to. If you versioned the project without catching up, these changes will not be included in the version. It is important to first catch up to changes made to the stream, retest with those changes and your soon-to-be-released changes, and then release your changes.

First taking the latest stream changes and retesting helps to ensure that the changes you are about to release will actually work with the current state of the stream.

24.9 Catching up your project

While you are working in the workspace, others may be releasing changes to the stream. To update the resources in your workspace, you need to catch up with the stream.

You control when to catch up with a stream and which incoming changes from a stream you want to accept into the workspace.

24.9.1 Incoming and outgoing changes

When you catch up to a stream a compare editor will be opened. In this you can see changes that have been released to the stream since you last caught up. As you accept these incoming changes, they are incorporated into your local workspace.

If your local workspace contains any outgoing changes that conflict with incoming changes from the stream, you must merge the conflicting changes in your local workspace as required.

Note: The stream itself is not changed when you catch up. When you accept incoming changes, these changes are applied to your workspace. The stream is only changed when you *release* your outgoing changes.

The Compare editor contains two panes.

The top one (Structure Compare pane) allows you to view the high-level structural differences between a stream resource and a local workspace resource.

The bottom one (Source Compare pane) allows you to see specific, line-by-line differences between a stream resource and a local workspace resource.

24.9.2 Dealing with conflicting changes

When catching up or releasing you may encounter conflicts.

A conflict occurs when you have locally modified a resource for which a more recent version is available in the stream. Specifically, the stream will contain a version newer than the base version of your resource. In this situation you can choose to do one of the following:

1. You can take the change from the stream (catch up to it) and discard your local changes.

This could be the right action to take if you have made unintended changes locally, or if you realize that the version in the stream is “better” than yours. Overwriting your local changes should be done with caution since you are in essence throwing work out.

2. You can *release* your change, overwriting the version in the stream.

   Note: This should be done only with great caution and, if possible, after consulting with the other developer(s). In essence you are throwing away someone else's work. The change you are overwriting may have other dependencies in the stream.

3. You can *merge* your work and the stream resource, by locally saving the merged resource. You may then later choose to release this merged result.

Merging will typically be the preferred option, because of the risk of unintentionally losing work associated with the other two options.

24.9.3 Merging changes

The Synchronize View shows those resources which are in conflict with the stream. For a given resource in conflict, typically you will want to merge your changes with changes in the stream's resource.

For example, lets assume that both you and another team member have modified the same html page.

Selecting that resource in the Synchronize view will display a comparison of the local resource and the stream version. By cycling through and merging the individual changes, you can decide for each change whether to accept the incoming change, reject it, or merge it with your local changes. When you are finished merging, you save your changes.

This overwrites your local resource with the result of the merge.

You can subsequently release this merged resource.

You can merge differences in the Catch Up/Release view on two levels.

- **Whole Document**: In the Structure Compare editor, select the resource that you want to merge so that the resource is displayed in the Text Compare editor. In the Text Compare editor, click the **Copy whole document from right to left** button to entirely replace the text in your local editor with the contents of the stream resource.

- **Current Change**: In the Text Compare editor, either use the **Select Next Change** and **Select Previous Change** buttons to navigate to the change that you want to merge, or simply click in either source pane somewhere in the desired change difference section. Click the **Copy current change from right to left** button when you want to overwrite the highlighted fragment with the corresponding modification in the stream. If you want to stick with the local resource, you don't need to do anything.

24.10 Versioning your project

Versioning a project saves the set of all resource versions in the project.The difference between versioning from the workspace or from the stream is in deciding which child resource versions should be part of the project version.

Version from workspace

When versioning a project from the workspace, the base version of the resources in the workspace are captured as part of the project version. This is the preferred method of versioning a project because you know exactly which resource versions will be in the project version.

This operation is allowed if you have outgoing changes or unreleased changes.

Unreleased changes are simply ignored and resources with outgoing changes can still have their base versions be part of the project version.

Versioning a project with unreleased or outgoing changes is handy if you have to split the project at the point where you started making changes to the resources and release the resources to another stream.

Version from stream

When versioning from the stream you are versioning the latest resource versions that are in the stream at that moment in time.

Note: You should not version your projects from the stream if you are not sure what is currently released in the stream.

In order to version a project, the following pre-requisite must be fulfilled:

- The project you want to version must be associated with a stream. If this is not the case, you must first associate it. See 24.5.3, "Associating a project with a stream" on page 516.

To version a project you need to do the following:

1. In the Navigator view, select the project that you want to version to the repository.
2. Select **Team—>Version from Workspace** from its context menu.
3. In the Version Selected Resources dialog,Figure 24-26, choose one of the following options to label the version of the selected project (or projects):
   - **Automatic**: The version label for each project will be automatically generated.
   - **One Name**: The version label for each project will be the one name that you specify.
   - **Name Each**: For each project, you will be prompted to enter its version label.



Figure 24-26 Version selected resource dialog

24.11 Releasing your project

As you make changes locally in your workbench, you are working isolated from the rest of the team. When you are ready to make your local resource changes available to other team members, you'll need to release your work to the stream. All such changes are referred to as *outgoing changes* when you do the synchronization.

Note: *Catch up first!* You should catch up to a stream before releasing to it. This ensures that you have the very latest work from the other team members.

After you have caught up with the stream, merged any conflicting changes in your local workbench, and tested your changes locally, you can go ahead and release your changes to the stream.

When you release changes to the stream, your changes are copied from your local workbench to the stream. As a result, these changes are then seen as *incoming changes* when other developers catch up to the stream later.

To release changes to a stream you have to:

1. Select the resource (or resources) that you want to release to the stream in the Navigator view.
2. Select **Team—>Synchronize with Stream** from the context menu. This will display the Synchronization view (Figure 24-27).
3. Click the **Release mode** button in the toolbar of the Synchronization view to filter out any modified workbench resources (outgoing changes) you may have.

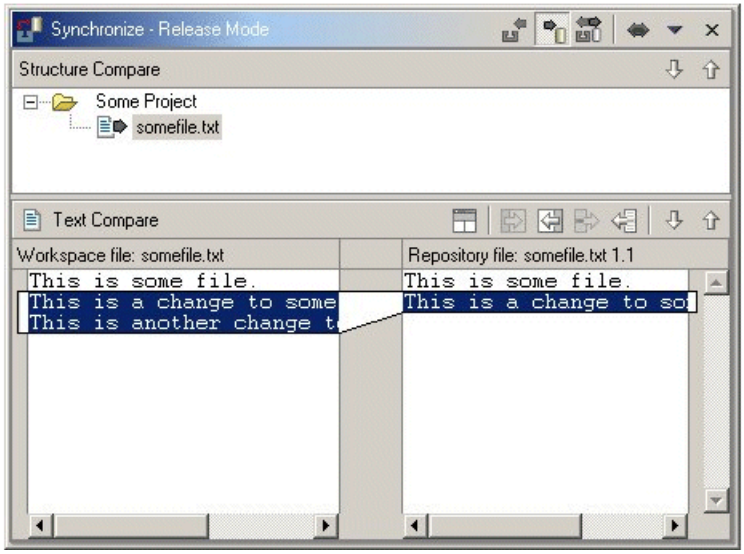


Figure 24-27 Synchronization dialog

4. If there are no conflicts (shown with red arrows), then you can release all the resources to the stream (go to Step 6). If there are conflicts, go to Step 5.
5. Resolve conflicts.

   A compare editor containing two panes will be opened.

   The top one (Structure Compare pane) allows you to view the high-level structural differences between a stream resource and a local workbench resource.

The bottom one (Source Compare pane) allows you to see specific, line-by-line differences between a stream resource and a local workbench resource.

The conflicts resolution process will likely involve a merge. Use the text compare area to merge resources that have conflicts.

You can copy changes from the repository version of the resource to the workbench version of the resource and save the merged workbench resource.

Once all the conflicts in the Structure Compare pane have been resolved, you are ready to release.

6. Select the top-most resource (or resources) in the hierarchy of the Structure Compare pane, bring up the context menu, and select **Release**.
7. In the Release Comment Needed dialog provide a meaningful comment for this release, for example "Fixed uninitialized variable in method x".
8. You are done when no elements remain in the Structure Compare pane.

24.12 Team development simulation

In this section, we will use a simple example to show you the main steps in the process of team development using CVS, and how Application Developer supports this process. You will use two separate workspaces to simulate branching and merging changes in a team environment.

24.12.1 Configuration

Create two local workspace directories, for example using the following commands from a Windows command prompt:

```
md C:/<Application Developer_ROOT>/developer1_ws
md C:/<Application Developer_ROOT>/developer2_ws
```

Then start Application Developer using these commands:

```
C:/<Application Developer_ROOT>/wsappdev
                -data C:/<Application Developer_ROOT/developer1_ws
C:/<Application Developer_ROOT/wsappdev
                -data C:/<Application Developer_ROOTT/developer2_ws
```

You should now have two instances of the IDE executing. Close the default perspective in each instance (to preserve memory) and open the team perspectives. Switch to the Repositories views.

In each instance of the IDE, create a connection to the local CVS repository installed in the previous chapter using the appropriate developers user ID and password, as described in 24.3, “Creating a CVS repository” on page 501.

Because we will be demonstrating the team capabilities using Java classes, also open the Java perspective in each instance.

24.12.2 Sequential development scenario

The simplest scenario to demonstrate is that of sequential development. This involves only one developer working on a file at the same time (Figure 24-28).

Both developers will be working on the same development stream with a class named `TeamTest1.java` in the `itso.wsad.team` package.

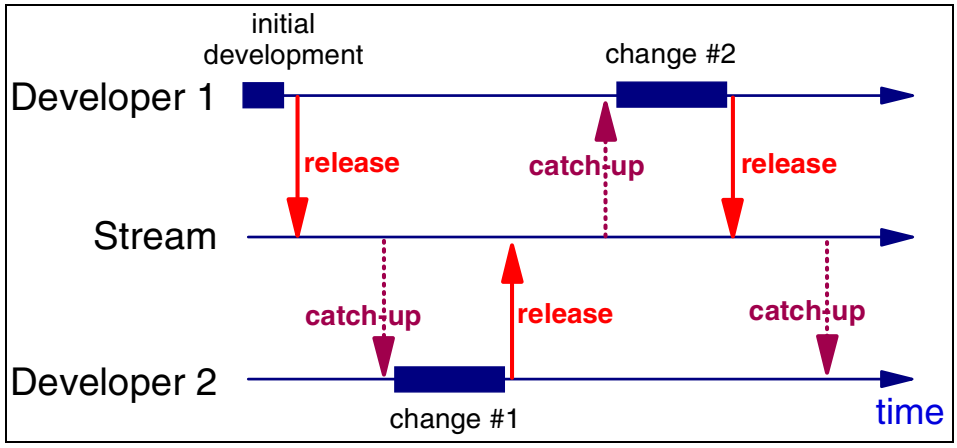


Figure 24-28 Sequential development scenario

Developer 1: Initial code release

Select the workspace for **Developer 1** and complete these tasks:

- Create a new Java project named `ITSOWSADTeam`.

> **Note:** The default location is now under the *developer1_ws* directory.

- Create a new package named `itso.wsad.team` under this project.
- Create a new Java class named `TeamTest1` inside this package. Accept all the default options.
- Close the editor that is open on `TestTest1`. Switch to the team perspective.

- Open the Properties dialog for the `ITSOWSADTeam` project. Switch to the **Team** category and click the **Change** button.
- Select the HEAD stream of the local repository connection and click **OK**.
- Click **OK** again to close the Properties dialog.

> **Note:** The project is now assigned to a stream.

- From the projects context menu, select **Team —> Synchronize with Stream**. This opens the Release Mode in the Synchronize view.
- Select the `ITSOWSADTeam` project entry in the Structure Compare panel and select **Release** from its context menu.
- Enter a comment of `Initial development complete` and click **OK**.

You have now completed the first step of the development. Next, Developer 2 must connect to the repository and retrieve the latest contents of the HEAD stream into his workspace.

Developer 2: Retrieve code and change

Switch to the **Developer 2** workspace and complete these tasks:

- Switch to the team perspective.
- Refresh the Repositories view. Expand the Stream and HEAD elements and the `ITSOWSADTeam` is displayed.
- Select the project entry and from its context menu select **Add to Workspace**.
- Switch back to the Java perspective. Open the `TeamTest1` class for editing.
- Add a new method with the following signature:

```
public void change1() { }
```

- Save the changes and close the editor.
- From the context menu for the item, select **Compare With —> Base version**. This allows us to view the changes made since the file was copied from the repository. Click **Done** when finished.
- Select the `ITSOWSADTeam` project. From its context menu select **Team—>Synchronize with Stream**. This opens the team perspective in Release Mode.
- In the Structure Compare panel, note only the `TeamTest1` file is shown in its hierarchy. Select the file and from its context menu click on **Release**.
- Provide a comment of `Change 1 complete` and click **OK**. No more entries should appear in the Synchronize view.

Developer 1: Catch-up and change

Developer 1 now makes a modification to the code. Switch back to that instance of the IDE and complete these tasks:

- Select the `ITSOWSADTeam` project. From its context menu select **Team—>Synchronize with stream**.
- The Synchronize view should open in Catch Up Mode. `TeamTest1` should be selected with in incoming change shown.
- From the context menu of TeamTest1, select the **Catch Up** menu.
- Switch back to the Java perspective. Open the `TeamTest1` class in the Java editor and add a second method with the following signature, save and close:

  ```
  public void change2() { }
  ```

- To release the changes to the stream again, select the `ITSOWSADTeam` projects **Team—>Synchronize with stream** menu.
- `TeamTest1` is again displayed with an outgoing change. Select the `ITSOWSADTeam` project in the Structure Compare view and select **Release**.
- Enter a comment of `Change 2 complete` and click **OK**.

Developer 2: Catch-up

Finally, refresh the source in the workspace of **Developer 2**:

- From the Java perspective, select the `ITSOWSADTeam` project. From its context menu select **Replace with—>Stream contents**.
- Open an editor on the `TeamTest1` class and notice both changes have been included in the file.
- To see a list of changes, select the **Team—>Show in Resource History** menu and the resource history window opens (Figure 24-29).

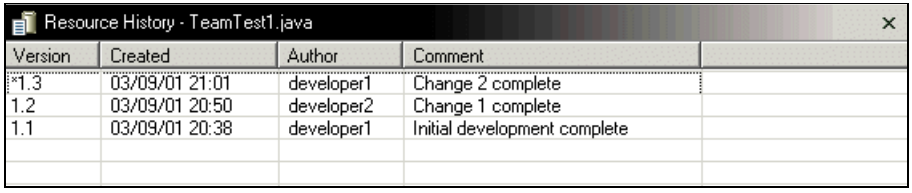


Figure 24-29 Resource history for sequential development scenario

Note the version numbering used here. Each time the file was released by a developer it increments from 1.1 to 1.2 then to 1.3.

An asterisk is displayed next to the *1.3* version because this is now the **base version** for the class that is in the workspace of Developer 2.

From the context menu of each of these elements in the table, the **Add to workspace** menu item is available.

Revert to previous version

Assuming you want to revert back to the previous version:

- Select the 1.2 version in the Resource History view. Select **Add to Workspace**.
- When prompted if you want to overwrite, select **Yes**.
- Change 2 should now be removed in the editor. To refresh the Resource History select **Team -> Show in Resource History** again. The view show an asterisk marking 1.2 as the base version.

Revert back to the latest changes by selecting the 1.3 version and adding it back to the workspace.

24.12.3 Parallel development in a single stream scenario

After we have demonstrated how to perform simple sequential development using the Application Developer team capabilities, the next step for you to understand is how to manage parallel development by two developers in the same stream. This scenario assumes that you have already completed the sequential scenario described in the previous section.

Developers 1 and 2 are both currently working on the `TeamTest1` class in the HEAD stream and both currently have version 1.3 as the base version in their workspace.

Let us now assume that they both make changes to the file simultaneously without letting each other know. This is illustrated in Figure 24-30.

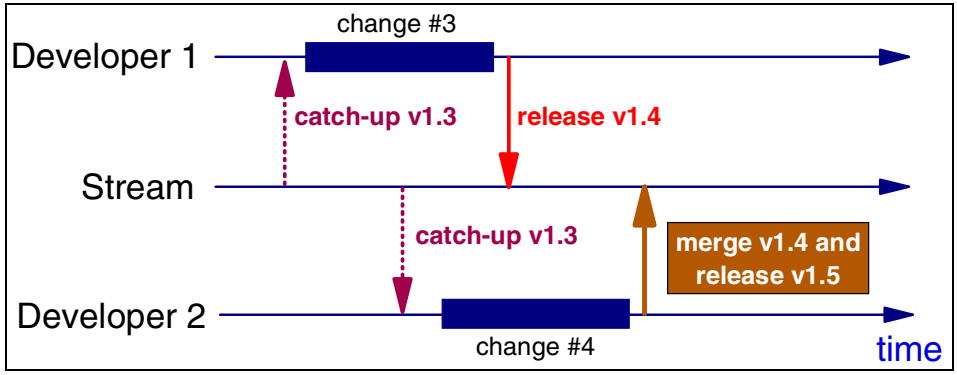


Figure 24-30 Parallel development in a single stream scenario

Developer 1: Change

Switch to the workspace for **Developer 1** and complete these tasks:

- Open the Java perspective and edit the `TeamTest1` class. Add a new method:

  ```
  public void change3() { }
  ```

- Save the changes and close the editor.
- Release the changes to the stream, adding a comment of `Change 3 complete`. By opening the Resource History view, you can see this was stored as version `1.4` in the repository.

Developer 2: Change and merge

Our **Developer 2** is unaware of the new version 1.4 and completes `change 4` on the local copy whose base version is still 1.3. Switch to the workspace of Developer 2 and perform this update:

- Open the Java perspective and edit the `TeamTest1` class. Add a new method:

  ```
  public void change 4() { }
  ```

- Save the changes and close the editor.
- Synchronize the `ITSOWSADTeam` with the HEAD stream. The Synchronize view should open in Catch Up Mode. `TeamTest1` is shown with a double-headed arrow icon  Figure 24-31. **This means there have been changes made to the stream, which conflict with the version Developer 2 wants to release**.

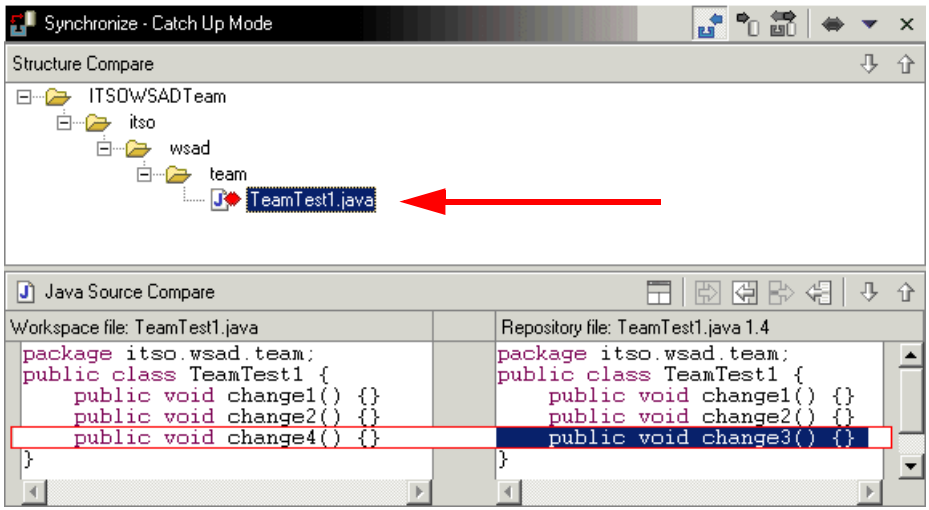


Figure 24-31 Conflicts shown in the Synchronize view

- To view what has happened, select the `TeamTest1.java` file from the Navigator view and launch the Resource History on the resource.
- We can see from the resource history that Developer 1 has made a change to the file since our base version.
- Next, from the Structure Compare panel in the Synchronize view, double-click on the `TeamTest1.java` file. This opens the Java structure comparison panel (Figure 24-32).

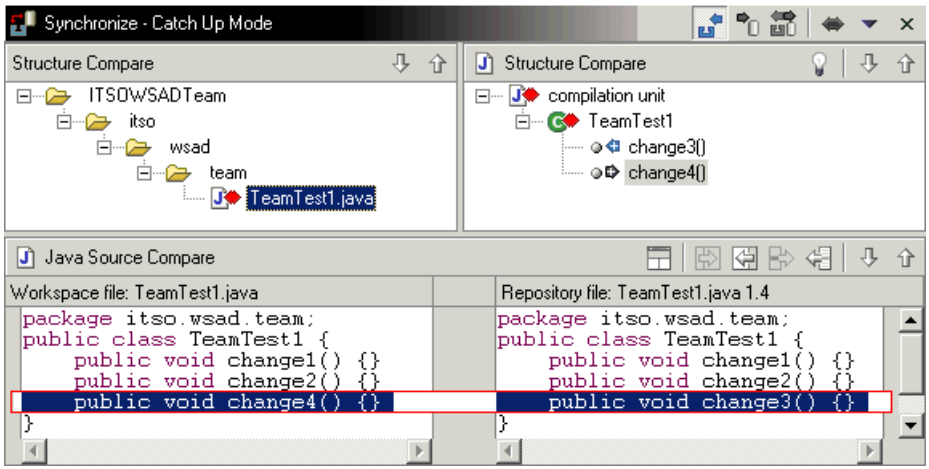


Figure 24-32 Displaying the Java structure comparison panel

- From this view, you can get a much better understanding of what has happened. `change3()` is shown with a left-arrow containing a **+** sign and `change4()` is shown with a right-arrow also containing a **+** sign.
- There are a number of icons in the Java sources compare panel (Figure 24-33). First click on **Control visibility of ancestor pane**.

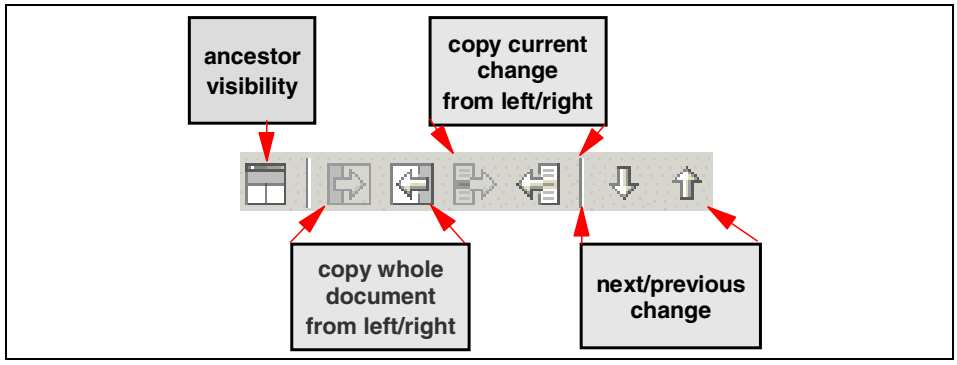


Figure 24-33 Icons in the Java sources compare panel

- This action opens a fifth panel in the Synchronize view. This window shows the source of the common ancestor of the two files—in this case version 1.3, which is the base version of the current code in the workspace (Figure 24-34).

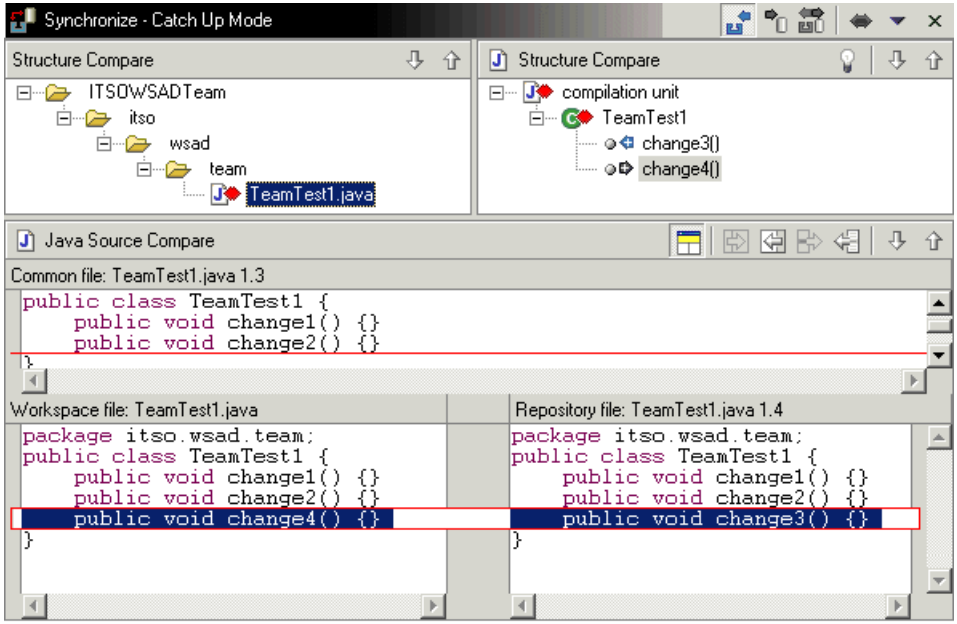


Figure 24-34 Displaying the common ancestor during comparison

- In this situation, merging the changes is relatively easy - you just have to add the `change3()` method into our workspace copy before releasing:
- Select the `change3()` item in the Java structure comparison panel in the top right of this view.
- Click on the **Copy current change from right to left** icon  .
- Right-click on the workspace entry in the bottom-left view and select **Save**.
- Save the workspace edition of the file. The Catch Up Mode in the Synchronize view should now be empty. Switch to the Release Mode.
- `TeamTest1` should now show that it contains both changes. Select the file and from its context menu select **Release**.
- Enter a comment of `Change 4 complete` and click **OK**.
- Open the resource history for the file and notice the new version is 1.5.

So what would have happened if you had not merged the changes?

By releasing the file, the current top of the stream would have been replaced with a version that does not include the `change3()` method, and this version would have been used for all future development.

Important: Streams do not support development branches—this must be done by creating a second stream.

24.12.4 Branching using multiple streams scenario

While building and maintaining applications, situations often occur where developers want to intentionally create a branch in the development process.

Some examples might include prototyping a new approach or performing maintenance on an old release.

In such circumstances, it is undesirable to release changes back into the HEAD stream immediately. This is the scenario you will focus on in the next section.

Figure 24-35 illustrates a scenario where the two developers branch the development of the application:

- Ensure both developers currently have version 1.5 loaded into their workspace.
- First, a project version is created from the current contents of the stream.
- Developer 2 then starts adding new functionality into the application while Developer 1 supports and maintains the version in production.

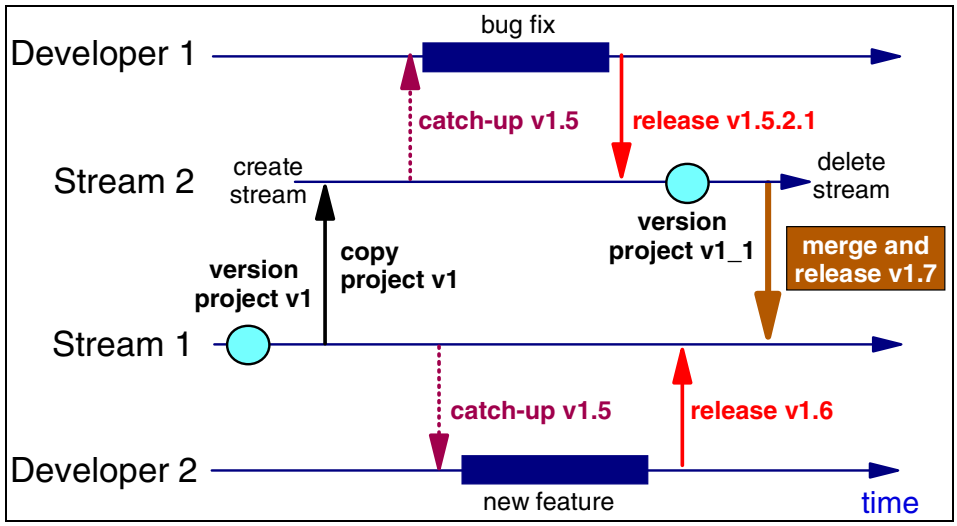


Figure 24-35 Parallel development using multiple streams scenario

Developer 1: Create new maintenance stream

Switch to the workbench for **Developer 1** and complete these steps:

- From the team perspective, open the Repositories view. Select the `ITSOWSADTeam` in the HEAD stream and click on **Version from Stream** from its context menu.
- Leave the selection on **Automatic** and click **OK**. The project gets version **v1**.
- Expand the project versions element in the tree. The `ITSOWSADTeam` element should have one entry contained inside it - `ITSOWSADTeam v1`. Expand the folders of the Java package - notice that it contains `TeamTest1.java` version 1.5.

> **Restriction:** It is only possible to create a new stream from a version of a project - not from the current elements at the top of another stream. **This ensures that there is always a baseline of the project we can revert to if we decide to remove all of the new streams contents**.

- Next, create a new stream for the local repository using the **File—>New—>Other** menu and selecting **Stream** from the **CVS** category. Enter `Maintenance` as the Stream name. Click **Finish**.
- From the Repositories view, select the new stream. Currently no projects are assigned to this stream. Select **Copy version to stream** from the context menu.

- In the Choose version dialog, select the `ITSOWSADTeam v1` project and click **OK**.
- Refresh the Repositories view. The project should now appear in the new stream.
- Select the project entry in the new stream and click on its **Add to workspace** menu item. If prompted to overwrite the existing files, select **Yes**. The workspace project is now assigned to the new stream.
- Perform the bug fix. In this case, add a new method in `TeamTest1` with the following signature:

```
public static void main bugfix1() { }
```

- Save the changes and close the editor.
- Release the changes to `TeamTest1` into the Maintenance stream.
- Add a comment of `Bug fix 1 completed`. Click **OK**. By viewing the Resource History for the file, you will see it has been added to the repository as version **1.5.2.1**.

Developer 2: Add feature to old stream

Next, switch to the **Developer 2** workspace and complete these tasks:

- Edit the `TeamTest1` class and add a new method whose signature is:

```
public void newfeature1() {}
```

- Save all changes and close the editor.
- Synchronize the project with the HEAD stream. Release the file with a comment of `New feature 1 added`. Click **OK**. This should now have created version **1.6** in the repository.
- Version the project as **v1_1** using the **Team—>Version from Workspace** menu from the projects context menu.

24.12.5 Merging streams

The final step is to merge the changes that were released into the Maintenance stream into the HEAD stream, so that the bug fix is included in the next release, which also contains the new feature.

Developer 2: Merge streams

In the **Developer 2** workspace, which still contains the project that is assigned to the HEAD stream, complete these tasks:

- Select the `ITSOWSADTeam` project. Click on the **Team—>Merge** menu from its context menu.

- In the first pane of the Merge wizard, select the local repository and the `ITSOWSADTeam` project. Click **Next**.
- Select as the Merge start point the version **v1** of the `ITSOWSADTeam` project. Click **Next**.
- Select as the end point the version **v1_1** of the `ITSOWSADTeam` project. Click **Finish**.
- The merge editor will open as shown in Figure 24-36.

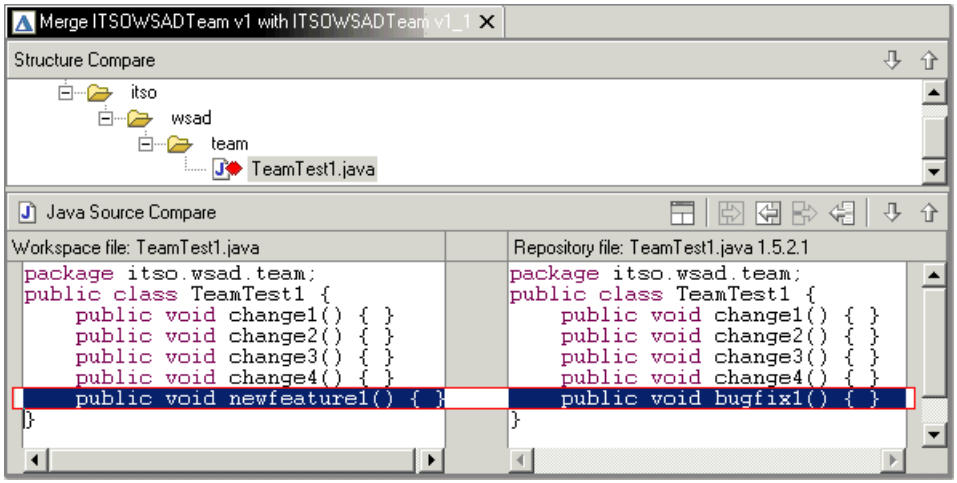


Figure 24-36 Merging the contents of two streams.

- A conflict is shown between the two streams: `newfeature1()` is in the HEAD stream and `bugfix1()` is in the Maintenance stream.
- Add the `bugfix1()` method to the version in the workspace. Save the changes in the merge editor.
- Perform a synchronization operation. Release the changes to `TeamTest1` and enter a comment of `Merged changes from maintenance`. Click **OK**.
- Open the Resource History view for the file. A new version**1.7** has been created in the repository containing the merged changes.

Developer 1: Delete maintenance stream

Finally, to clean up, switch back to the Developer 1 workbench and delete the Maintenance stream.

24.13 Additional team topics

The following sections discuss some advanced topics relating to team development with Application Developer and CVS.

24.13.1 Determining which files are managed

There are two mechanisms you can use to determine which files are managed by the repository:

- The global **Ignore** facility, provided by the workbench.

  To add a global ignore pattern for the workbench, simply select the **Window—>Preferences** to open the Workbench preferences dialog and select the **Team—>Ignored Resources** category. Click **Add** to add the pattern to ignore.

- The **CVS Ignore** facility, provided by a file called .cvsignore in the file system

  The .cvsignore file should be added to the workspace directory of the project whose files you want to ignore. A good example would be to ignore all of the files stored in the classes folder of our project. In this case, create a .cvsignore file in the YourProject/webApplication/WEB-INF directory containing the line:

  ```
  classes
  ```

  Note: The ignore file only relates to the files and folders contained in that directory, so you may require many `.cvsignore` files throughout the project.

  To share the ignores with the other developers in the team, simply release the `.cvsignore` file into the stream.

A good practice is to only include source files in the project. For example, in the case of EJBs, store the generated types in the repository along with the generated metadata such as ejb-jar.xml, but do not store the JAR files or compiled classes.

24.13.2 Backing up the CVS repository

Probably the most important task for a repository administrator is to ensure that the CVS repository is backed up on at least a daily basis.

Unlike VisualAge for Java's team repository, which required the administrator to create a copy of the locked repository file before the backup, the CVS repository directory structure can be backed up from its current location.

To restore from a backup, simply restore the files from the backup copy into their previous directory structure. If there are no backup facilities available, create a scheduled batch script that creates a timestamped zip file of the entire CVS directory and copies it to another machine.

24.13.3 Repository management

When working with CVS, it is generally recommended not to archive and remove old versions of projects from the repository.

Note: This is one of the major limitations of the product.

There is a feature in the command line tools installed with CVS, invoked by:

```
cvs admin -o
```

This command can remove either a specific version number of a range of versions.

Important: Extreme caution is recommended as such changes cannot be backed out.

24.13.4 Implementing security

In the current installation, all valid users of the Windows server have unlimited access to read and write files in the VS repository. This is unlikely to be a satisfactory scenario.

The creation of two files is all that is required to implement tighter security for an individual repository. On the server, create two text files called **readers** and **writers** in the root directory of the CVS repository.

These files should contain a new line-separated list of users that you want to grant that level of permission. If no writers file is supplied, then the CVS server implicitly assumes that all users have both read and write permission. If only the writers file exists, CVS grants all users read permission and only the specified users write permission.

Note: You should not include the same user in both files - only in the file with the higher permission level you want to provide. Putting the same name in both readers and writers assumes that you want to only give read access.

Unfortunately, there is no facility in CVS to limit which users have to capability of versioning projects or creating new streams.

Tip: Rational ClearCase may be a more suitable alternative to CVS if an organization has these requirements.

24.13.5 Build scripts

Because CVS has a command line interface, it is easy to develop a batch client that retrieves the latest versions from a particular stream to a directory structure using `cvs update -q`.

A series of command line tools can then be invoked, such as the Java compiler, WebSphere command line tools (WSCP, XMLConfig) or Apache Jakarta Ant scripts to build and deploy an application either to a test server, or even to automate the process of releasing a build.

These topics are outside the scope of this document, but are documented in some detail in the *WebSphere Version 4 Application Development Handbook*, SG24-6134.

24.13.6 Managing class paths

One of the problems you can often ran into is how to do team development with CVS when each developer has their workspace in a different directory or drive. A number of the files which are included in the repository by default include metadata containing drive and directory names.

While the simplest solution is to ensure everyone is working from an identical workspace path, there are some workarounds for the following files:

- **.classpath**: This file contains the class path definition for the project. It is easier to define a Application Developer class path variable such as WSADHOME and WSHOME to store the root directory of the installation and the current workspace the developer is using on each developer workstation, and then use these to refer to the path of specific JAR files used in the build path.
- **server-cfg.xml**: This is the file likely to cause the most problems, because it contains a number of references to class paths for JDBC drivers, and the WebSphere transaction log file, which may be different for each client. We recommend that if developers have different workspace configurations, do not version Server projects but redefine them manually on each developer desktop with the appropriate class paths and JDBC drivers.

24.13.7 Watching a file for changes

Another common requirement by previous VisualAge for Java users might be to be informed of changes made to a resource by other developers automatically when a release is performed.

CVS provides only a limited support for this mode of operation, however, it must be performed using a command line CVS client.

Once the client has been installed, and a connection established to the repository, you can enter this command:

```
cvs watch add -a commit files
```

This command enables the default notification mechanism to inform the developer that a commit (release) has been performed on the file, usually through e-mail. To remove watches on the files, you have to enter the command:

```
cvs watch remove -a commit files
```

And finally, to find out who else is watching those files, use:

```
cvs watchers files
```

24.13.8 Other CVS commands

For a comprehensive list of the extensive CVS commands, browse the on-line documentation available at:

http://www.cvshome.org

25

Using Rational ClearCase

This chapter describes the following:

- Rational ClearCase
- Installing ClearCase LT Server and Client
- Rational ClearCase terminology
- ClearCase integration with Application Developer
- ClearCase tutorial

© Copyright IBM Corp. 2002

25.1 What is Rational ClearCase?

Rational ClearCase is a Software Configuration Management (SCM) product that helps to automate the tasks required to write, release, and maintain software code.

Rational ClearCase offers the essential functions of version control, workspace management, process configurability, and build management. By automating many of the necessary and error-prone tasks associated with software development, Rational ClearCase helps teams of all sizes build high quality software.

ClearCase incorporates Unified Change Management (UCM), Rational's "best practices" process for managing change at the activity level and controlling workflow.

UCM is easy-to-adopt and can be applied to projects “out-of-the-box”, enabling teams to get up and running quickly. However, it can be replaced with any other process that you already have in place at your site.

Rational ClearCase provides support for parallel development. With *automatic branching* and *snapshot views,* it enables multiple developers to design, code, test and enhance software from a common code base.

Snapshot views support a *disconnected use* model for working away from the office. All changes since the last snapshot are automatically updated once you are connected again.

Rational ClearCase's *diff/merge technology* makes it practical to merge source code, HTML and XML. It automatically accepts uncontested changes and highlights conflicts for fast resolution without manual intervention.

Rational offers two version of its ClearCase product - *ClearCase* and *ClearCase LT*. ClearCase LT is a ‘light’ version for support of small teams that do not need the full functionality of the complete ClearCase product (distributed servers, database replication, advanced build management or transparent file access, and so on). For the ‘full-sized’ ClearCase, Rational also provides an add-on *MultiSite* feature.

Attention: Note that the full version of ClearCase is *not* included with Application Developer.

25.1.1 ClearCase highlights

Rational ClearCase family of products has these highlights:

- Offers version control, workspace management, build management and process configurability.
- Versions all development artifacts.
- Enables nonstop parallel development — even across geographically distributed sites.
- Provides transparent workspaces for global data access.
- Integrates with Rational ClearQuest to provide a seamless approach to defect and change tracking.
- Enables Unified Change Management — Rational's activity-based process for managing change.
- Scales from small project teams to the global enterprise.
- Ships with Rational Suite for a complete change process across the life cycle.
- Offers process configurability without expensive customization.
- Provides advanced build auditing.
- Provides Web interface for universal data access.
- Features graphical interface for easier focus on priority tasks.
- Integrates with leading IDEs and development tools as well as Web development and authoring tools.

25.1.2 ClearCase LT

ClearCase LT has these features:

- Is an entry-level version control tool for small project workgroups.
- Is suited for teams with projects that don't require distributed servers, database replication, advanced build management or transparent file access.
- Two main factors make it easy to install and use:

  a. It is the only entry-level version control tool that offers an out-of-the-box process for automating the process of software change and controlling workflow at the activity level.

     UCM saves you the time it would take to configure your own processes.

     It provides an *activity-based framework* that automatically associates an activity with a change set including all related software artifacts, UCM enables all team members to create, view, schedule and integrate work in a more familiar context.

b. It offers features specifically designed to get you up and running quickly. These include the *ClearCase Explorer,* an intuitive, graphic interface enabling team members to clearly focus on high-priority tasks. The ClearCase Explorer provides easy-to-customize shortcuts, which makes managing multiple work spaces a snap.

   Rational ClearCase LT also provides a server start-up wizard that speeds UCM adoption. It can graphically import data from competitive version control tools, including Merant (PVCS).

25.1.3 ClearCase and ClearCase MultiSite

As your project grows it may require advanced features such as distributed servers, database replication, advanced build management and transparent file access. When required, you can seamlessly upgrade to the one of the full-featured Rational ClearCase product without having to retrain your team or change your processes or tools.

This version is *not* included within Application Developer package.

The tables below will help you identify which Rational ClearCase solution is right for you. The first (Table 25-1) summarizes the key capabilities which each Rational ClearCase solution offers to various types of software development teams. The second (Table 25-2) provides a feature list comparing Rational ClearCase and Rational ClearCase LT capabilities.

Table 25-1 Key capabilities of ClearCase

ClearCase LT	ClearCase	ClearCase MultiSite
Small, independent workgroups	Medium to large teams	Geographically distributed teams
Snapshot views	Dynamic and snapshot views	Add-on for Rational ClearCase only
Single server	Multi-server	Multi-site data replication

Table 25-2 Comparison of ClearCase and ClearCase LT features

Feature	ClearCase LT	ClearCase
Activity based change process	**X**	**X**
UCM out of the box process	**X**	**X**
Windows Explorer integration	**X**	**X**
Web browser interface	**X**	**X**

Feature	ClearCase LT	ClearCase
Version management	X	X
Promotion model	X	X
Parallel development	X	X
Life cycle tool integration	X	X
Snapshot" or sandbox views	X	X
Windows/UNIX inter operability	X	X
Compare baselines by activity or file	X	X
Clearmake tool	X	X
COM API (CAL runtime on Windows)	X	X
COM API development support		X
Detailed build auditing		X
Binary sharing		X
Build dependency automation		X
Distributed builds (UNIX only)		X
Parallel builds (Windows/UNIX)		X
Distributed servers		X
Transparent, "dynamic" views		X
Network attached storage devices		X
MultiSite server replication add-on		X

For more information see http://www.rational.com/products/clearcase.

25.2 Installing ClearCase LT

The following sections describe how to install the ClearCase LT Server and Client.

25.2.1 Installing ClearCase LT Server

Rational ClearCase LT (both the client and the server) is included with your Application Developer license.

Instructions on how to install Rational ClearCase LT are provided in the cc_install.pdf file, which is located in the Rational ClearCase LT sub-directory.

To start the installation, run `setup.exe` from the Rational ClearCase LT directory (Figure 25-1).

Important: To be able to use the Rational ClearCase LT license that comes with Application Developer, you must install Rational ClearCase LT on the same machine as Application Developer. It will not run properly otherwise. When installing the product you can ignore any references to obtaining a license, as it is provided by Application Developer.

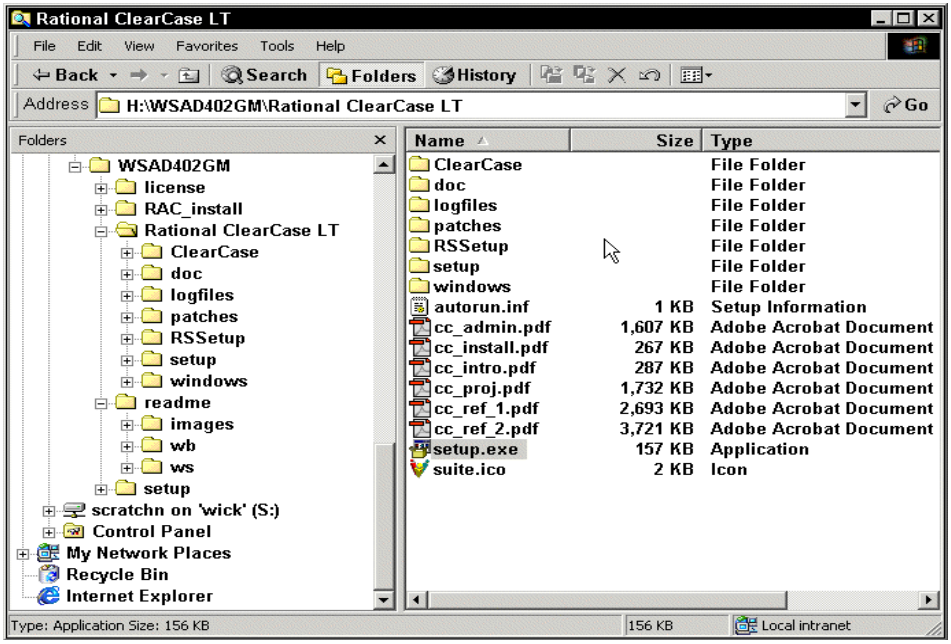


Figure 25-1 Running setup.exe to install ClearCase LT

If you already have ClearCase installed, be sure to apply the November 2001 or later patch from Rational. If you want use ClearCase LT, you should install the version of it provided with Application Developer since this version already has the latest patches applied.

When you have installed the ClearCase LT server, review the `Rational ClearCase: Upgrades, Patches and Service Releases` page on the Rational Web site and make sure that the latest fixes have been applied.

25.2.2 Installing the ClearCase LT client

During the Application Developer installation, the Version Control Interface plug-in you select will be enabled. You should have selected Rational ClearCase as the Version Control Interface (Figure 25-2). If you did so, your ClearCase LT client will already be installed.

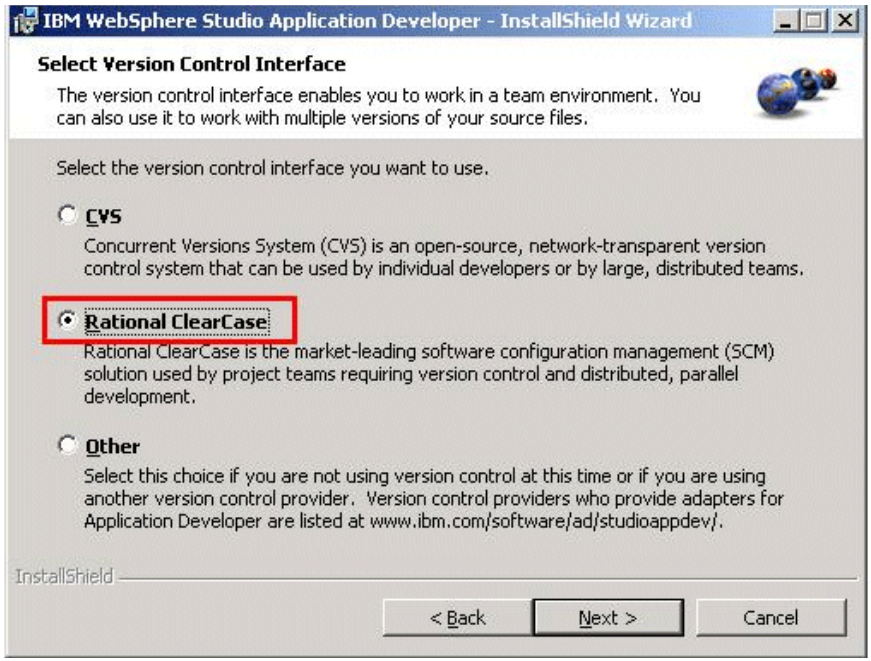

IBM WebSphere Studio Application Developer - InstallShield Wizard
Select Version Control Interface
The version control interface enables you to work in a team environment. You can also use it to work with multiple versions of your source files.
Select the version control interface you want to use.
CVS
Concurrent Versions System (CVS) is an open-source, network-transparent version control system that can be used by individual developers or by large, distributed teams.
Rational ClearCase
Rational ClearCase is the market-leading software configuration management (SCM) solution used by project teams requiring version control and distributed, parallel development.
Other
Select this choice if you are not using version control at this time or if you are using another version control provider. Version control providers who provide adapters for Application Developer are listed at www.ibm.com/software/ad/studioappdev/.
InstallShield
< Back
Next >
Cancel

Figure 25-2 Select Version Control Interface pane of the Install Shield Wizard

If you have already installed Application Developer, but you are not sure which version control interface was installed, execute the `cm_status.bat` file that is located in the directory, `<WSAD_ROOT>\plugins\com.rational.clearcase`.

If ClearCase was selected during the installation, you should see results similar to those in Figure 25-3.

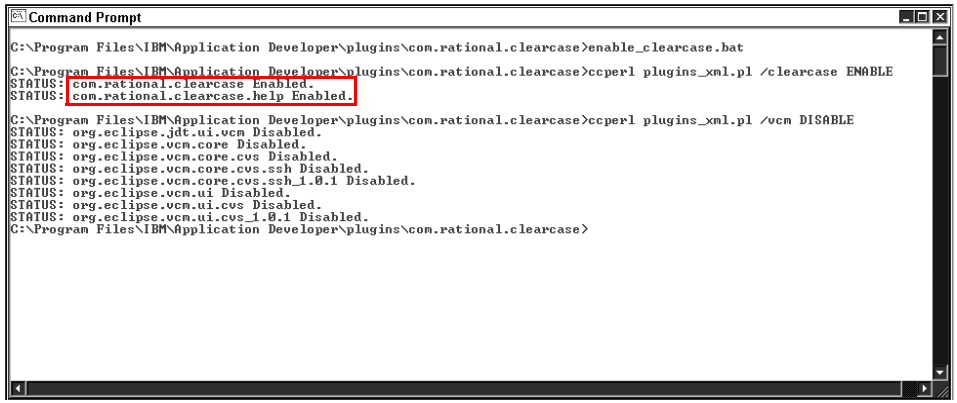


Figure 25-3 Command window showing that ClearCase is enabled

If you instead selected **CVS** when you installed Application Developer you will see results similar to those in Figure 25-4:

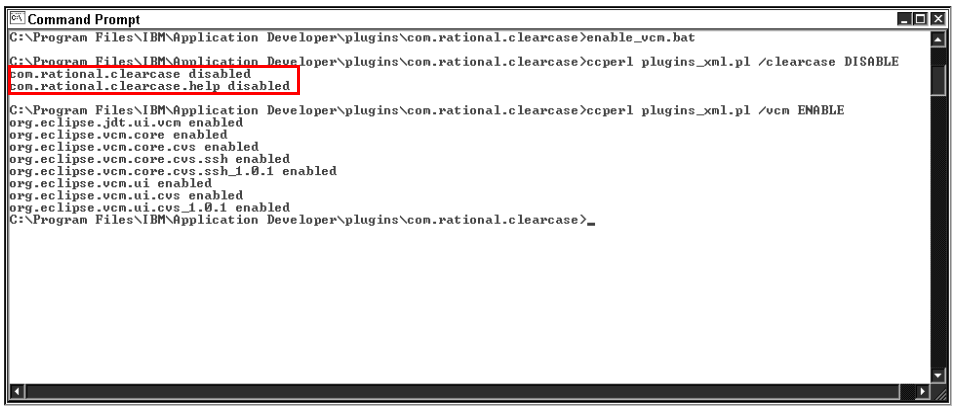


Figure 25-4 Command window displaying that ClearCase is not enabled

If you want to change the version control interface to ClearCase, you can do this by uninstalling and reinstalling Application Developer.

There is however an easier way to switch version control interface. You can execute the `enable_clearcase.bat` file that is located in the directory, <WSAD_ROOT>plugins\com.rational.clearcase. Close down Application Developer and run this file. This will disable the current CVS interface and enable ClearCase LT.

(In the same directory, you will also find the `enable_vcm.bat` file that does the opposite: disables ClearCase LT and enables CVS.)

Tip: To avoid surprises, *delete and re-create your workspace* after changing the version control interface.

25.2.3 Learning ClearCase LT

Before you begin working with ClearCase LT, it is a good idea to first take a look at the ClearCase tutorial. From your Windows **Start** menu, select **Programs—>Rational ClearCase LT Server—>Tutorial** (Figure 25-5).

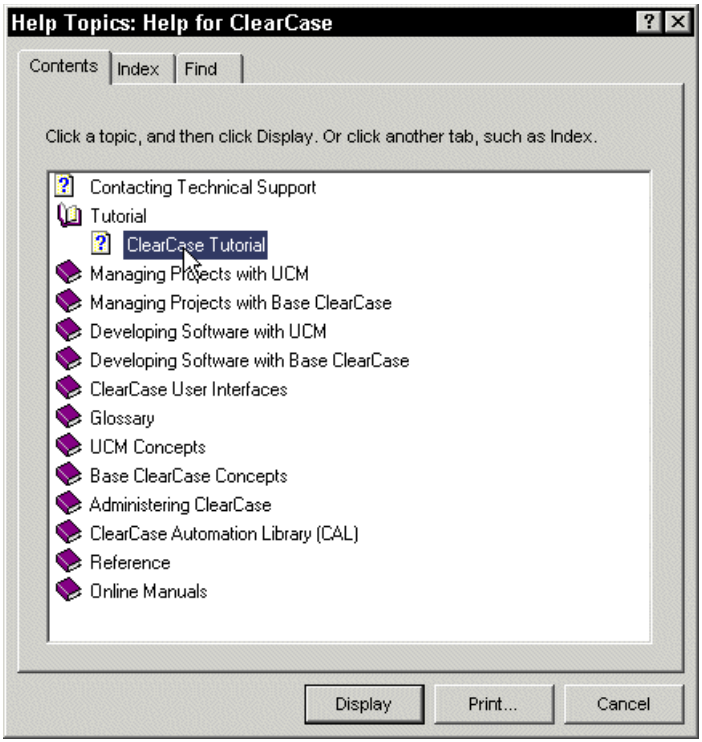


Figure 25-5 Starting the ClearCase tutorial

Take some time to review this tutorial. It does not demonstrate how to integrate ClearCase LT with Application Developer, but it will give you an idea of how to work with ClearCase LT itself. This is necessary to understand how ClearCase LT integrates with Application Developer.

25.3 Basic ClearCase terminology

- **UCM**: *Unified Change Management* - An out-of-the-box configuration management process, layered on ClearCase and ClearQuest functionality, for organizing software development teams and their products. Members of a project team use *activities* and *components* to organize their work.
- **Project**: A ClearCase (UCM) entity that contains the configuration information needed to manage a significant development effort, such as a product release. Use the project to set policies that govern how developers access and update the set of files and directories used in the development effort.

  A project includes one integration stream, which configures views that select the latest versions of the project's shared elements, and typically multiple development streams, which configure views that allow developers to work in isolation from the rest of the project team.
- **Component**: A ClearCase entity that you use to group a set of related directory and file elements within an (UCM) project. Typically, you develop, integrate, and release the elements that make up a component together. A project must contain at least one component, and it can contain multiple components. Projects can share components.
- **Activity**: A ClearCase UCM entity that tracks the work required to complete a development task. An activity includes a text headline, which describes the task, and a change set, which identifies all versions that you create or modify while working on the activity. When you work on a version, you must associate that version with an activity.

  If your project is configured to use the UCM-ClearQuest integration, a corresponding ClearQuest record stores additional activity information, such as the state and owner of the activity.
- **VOB**: **Versioned object base -** The permanent data repository in which you store files, directories, and metadata. Files and directories under ClearCase control are called *elements*, and each checked-in revision of an element is called a *version*. Typically, a VOB contains all of the different versions of an element and the metadata such as labels and checkout comments used to describe each version.
- **View**: Provides a directory tree containing one version of each file in your project. In the view, you modify source files, compile them into object modules for testing purposes, and format them into documents.
- **View-tag**: A unique, descriptive name for each view. In ClearCase, the view-tag must be unique within a given network region. Choose a name that helps you determine the owner and purpose of the view. Generic names can lead to confusion. A view-tag must be a simple name (that is, it must follow

the format of a single file or directory name with no special characters or spaces).

- **Baseline**: A ClearCase UCM entity that typically represents a stable configuration for one or more components. A baseline identifies activities and one version of every element visible in one or more components. You can create a development *stream* or *rebase* an existing development stream from a baseline.
- **Stream**: A long-lived ClearCase entity. It is a member of a single UCM project and is a mechanism for creating and recording configurations. A stream identifies the exact set of versions currently available for you to view, modify, or build. UCM uses baselines and activities to describe a stream's configuration.

  When you create a stream, its original configuration is the same as a baseline. That is, it contains a single version of each element in a component. When you modify a stream's configuration, you assign the modifications to one or more activities. Hence, **a stream's configuration is a given baseline plus one or more activities**.
- **Rebase**: A ClearCase (UCM) operation that makes your development work area current with the set of versions represented by a more recent baseline in the integration stream.
- **Check in/Check out**: ClearCase works with Check in/Check out concept. You check out a resource to work on it and check it in when the changes are complete.

  Checking in a resource creates a new version of it in the VOB. In some version-control systems, only one person at a time can reserve the right to create a new version. In other systems, many users can compete to create the same new version.

  ClearCase supports both models by allowing two kinds of **checkouts**: **reserved** and **unreserved**.
- **Checkouts in server mode**: When you check out a file in server mode, the checkout is reserved. A reserved checkout gives you the exclusive right to check in the file, which creates the next version. Only one checkout can be reserved.
- **Checkouts in local mode**: When you check out a file in local mode, the checkout is unreserved. More than one person can check out the file at the same time, and no one is guaranteed the right to perform the next check in.

  If several people checked out a page as unreserved in their local views, the first person to check in the page creates the new version. Everyone else must merge the latest version into their own work before they can check in their changes.

In local mode, you can change a reserved checkout to unreserved and vice versa by using the ClearCase menu.

25.4 ClearCase and Application Developer

The following sections will explain how to use ClearCase LT as a version control system for Application Developer.

25.4.1 ClearCase help in Application Developer

Application Developer provides some documentation for using Rational ClearCase. The online help details the options in Application Developer that are relevant to Rational ClearCase.

To access the help documentation, start Application Developer and select **Help—>Help Contents** to open the Help perspective. Select **Rational ClearCase** from the help domains combo box (Figure 25-6).

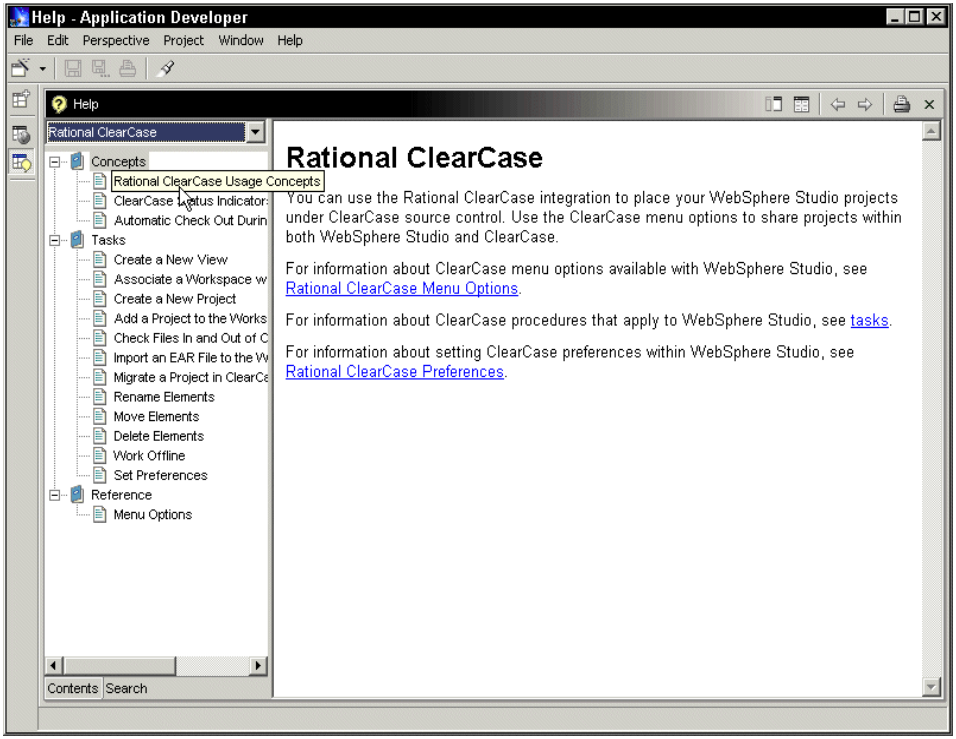

Help - Application Developer
File Edit Perspective Project Window Help
Help
Rational ClearCase
Concepts
Rational ClearCase Usage Concepts
ClearCase tus Indicator:
Automatic Check Out Durin
Tasks
Create a New View
Associate a Workspace w
Create a New Project
Add a Project to the Works
Check Files In and Out of C
Import an EAR File to the W
Migrate a Project in ClearCa
Rename Elements
Move Elements
Delete Elements
Work Offline
Set Preferences
Reference
Menu Options
Contents Search
Rational ClearCase
You can use the Rational ClearCase integration to place your WebSphere Studio projects under ClearCase source control. Use the ClearCase menu options to share projects within both WebSphere Studio and ClearCase.
For information about ClearCase menu options available with WebSphere Studio, see Rational ClearCase Menu Options.
For information about ClearCase procedures that apply to WebSphere Studio, see tasks.
For information about setting ClearCase preferences within WebSphere Studio, see Rational ClearCase Preferences.

Figure 25-6 Using the ClearCase online help

25.4.2 Rational ClearCase preferences settings

You can change the Rational ClearCase preferences and configuration options within Application Developer to suit your needs. To view and change these options, select **Window—> Preferences**. In the Preference dialog, select **Rational ClearCase** to set the configuration options (Figure 25-7).

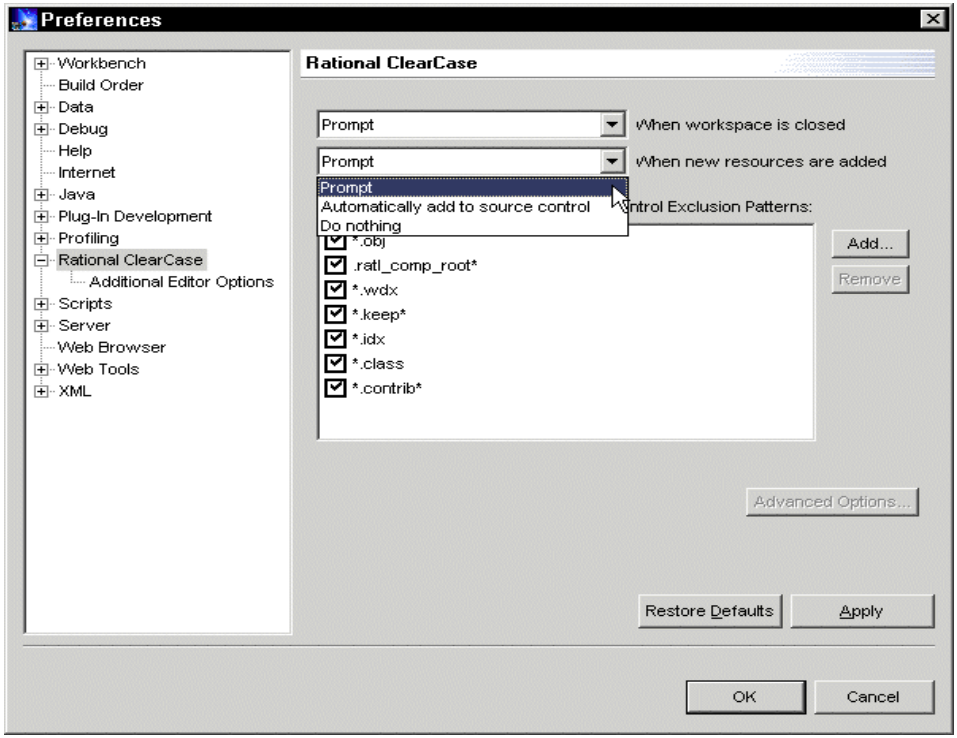


Figure 25-7 ClearCase Preferences dialog

25.5 Using ClearCase with Application Developer

In this chapter, we will step through an example of using Rational ClearCase with Application Developer.

To keep the example simple, we are assuming that you are using one Windows workstation with both the ClearCase server and client installed.

Once you are familiar with the basic activities, you will be able to set up a “real-world” developer's environment where the server is installed on another server and/or another operating system.

Some activities described in the example must be performed by the ClearCase administrator, while others can be done by the developers.

ClearCase is fully integrated into Application Developer as a plug-in and you can work with it directly from within your Application Developer workspace. You also still have access to all the functionality offered by the native ClearCase environment. Some tasks need to be performed in that environment.

25.5.1 Create a new ClearCase VOB

A ClearCase Versioned Object Base (VOB) will have been created during the server installation, but to help you get familiar with the process of creating one we will walk through the creation process.

Creating a VOB must be done in the ClearCase server. To perform this task, select **Programs—>Rational ClearCase LT Server—>Create VOB**. The VOB creation wizard will be displayed (Figure 25-8).

You must enter the VOB-tag, in our example `WSAD_VOB1`. This will be the registered name for the new VOB. Whenever you perform operations on this VOB you will always refer to this VOB-tag.

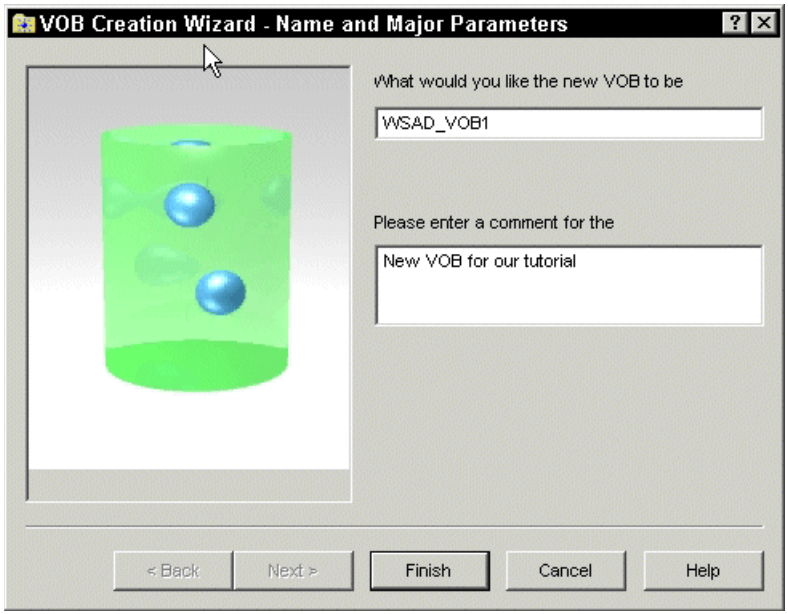

VOB Creation Wizard - Name and Major Parameters
What would you like the new VOB to be
WSAD_VOB1
Please enter a comment for the
New VOB for our tutorial
< Back
Next >
Finish
Cancel
Help

Figure 25-8 Creating a new ClearCase VOB

When you click **Finish** you will first be prompted for a confirmation, then a progress window will be displayed while the task is running. After it has completed you will be shown a summary of the task. You can close this window.

Since we are using the standalone workstation, the VOB will be created on your local machine. After the command has executed you should see the VOB added to the file hierarchy on your workstation (Figure 25-9).

This task is usually executed only once during the installation process and it is done by the Administrator.

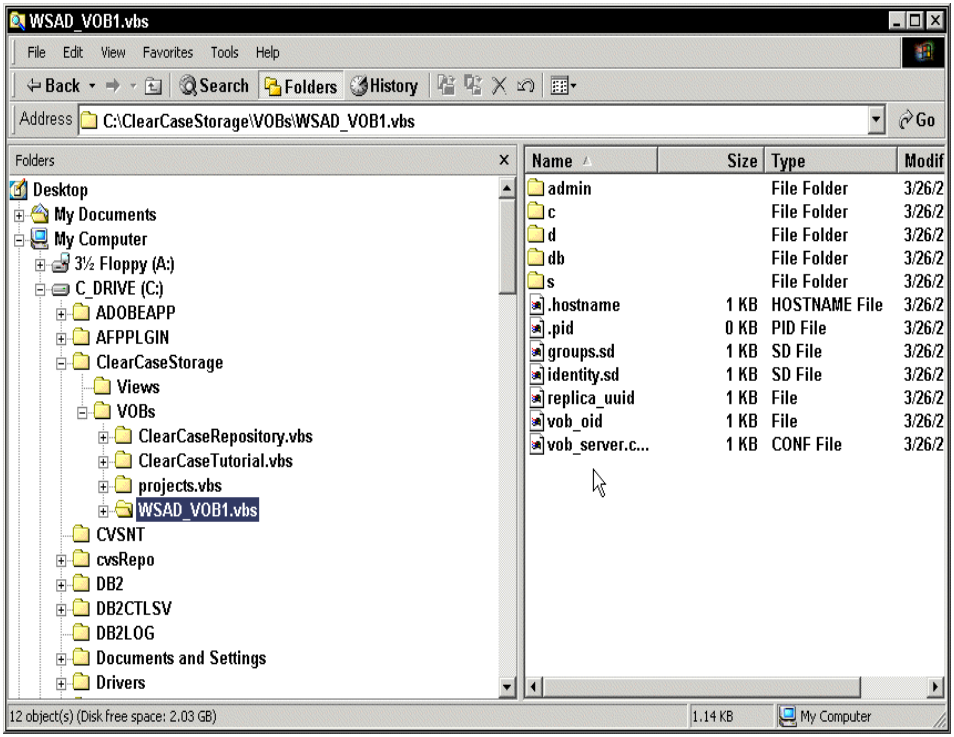


Figure 25-9 The new VOB created under ClearCase Storage/VOBs directory

25.5.2 Associate with a ClearCase view

The term *workspace* has different meanings in Application Developer and in ClearCase:

- In **Application Developer** a workspace is the set of projects a user can work within a specific instance of Application Developer.

- In **ClearCase** a workspace generally means a Rational ClearCase view. A view provides controlled, versioned access to the exact set of file versions in the VOB.

> **Tip:** Before you start adding the Application Developer projects to the ClearCase views, we generally recommend you to maintain a separate Application Developer workspace for each Rational ClearCase view you use.

To associate a Application Developer workspace with a ClearCase view, complete the following steps:

1. Copy the Application Developer workspace directory and its contents that you want to have under ClearCase control to another location. Make sure to keep it under the directory where Application Developer is installed. In this example, we will copy it from c:\WSAD_ROOT\workspace to c:\WSAD_ROOT\workspaceRedbook.

2. Create a new Application Developer shortcut on your desktop to point to this newly created workspace. If you already have a Application Developer shortcut you can do this by copying it and changing its properties to use the `-data` flag, which takes as its argument the path to the new directory where the workspace is stored. In our example, it should state: `\WSAD_ROOT\wsappdev.exe -data c:\WSAD_ROOT\workspaceRedbook.`

   Start Application Developer and delete the projects that you don't need.

3. Connect to ClearCase. Rational ClearCase allows you to work online or offline. If you need to access the ClearCase server you must first connect to it. Normally you only need to connect once to start your task activities.

   To connect to the ClearCase server from within Application Developer (using the J2EE perspective as an example), select the **ClearCase—>Connect to Rational ClearCase** menu option or click the **Connect to Rational ClearCase** icon  (Figure 25-10).

   The Connect option is enabled when you have started Application Developer. After you have connected this option becomes inactive and remains that way for the duration of your Application Developer session, while the other ClearCase options become enabled.

   > **Note:** You may need to select a project first to see the ClearCase options enabled.

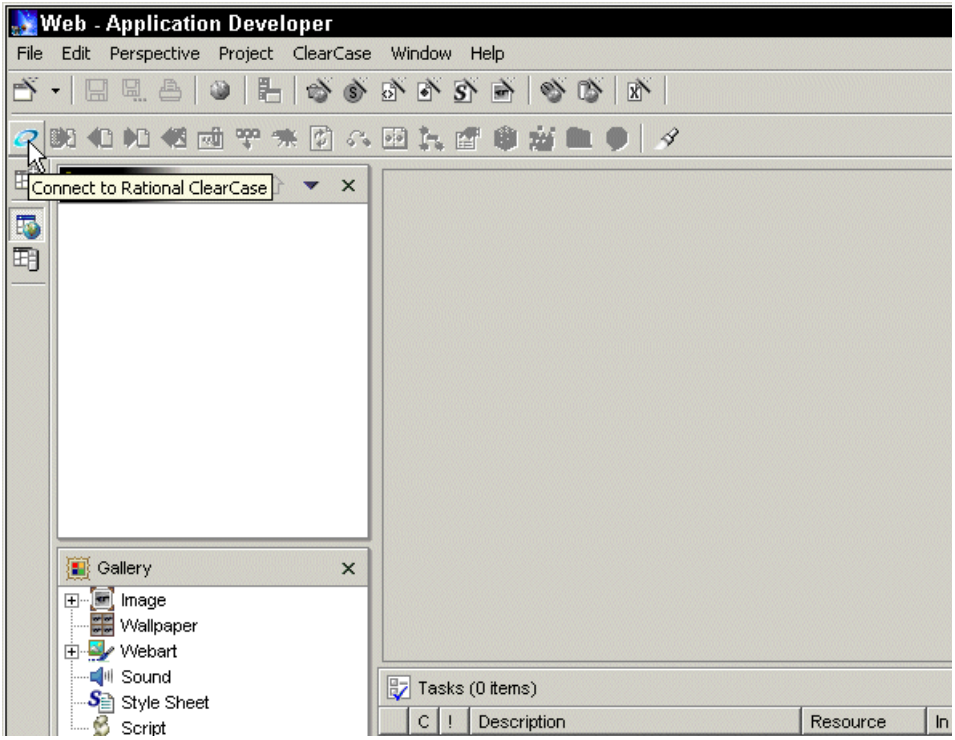


Figure 25-10 Connecting to ClearCase server

4. Create a new view. You now need to create a ClearCase view and connect it to the associated Application Developer workspace. Again this needs to be done only once and is an Administrator task. To create the new view, choose **ClearCase—>Create New View** from within Application Developer (Figure 25-11).

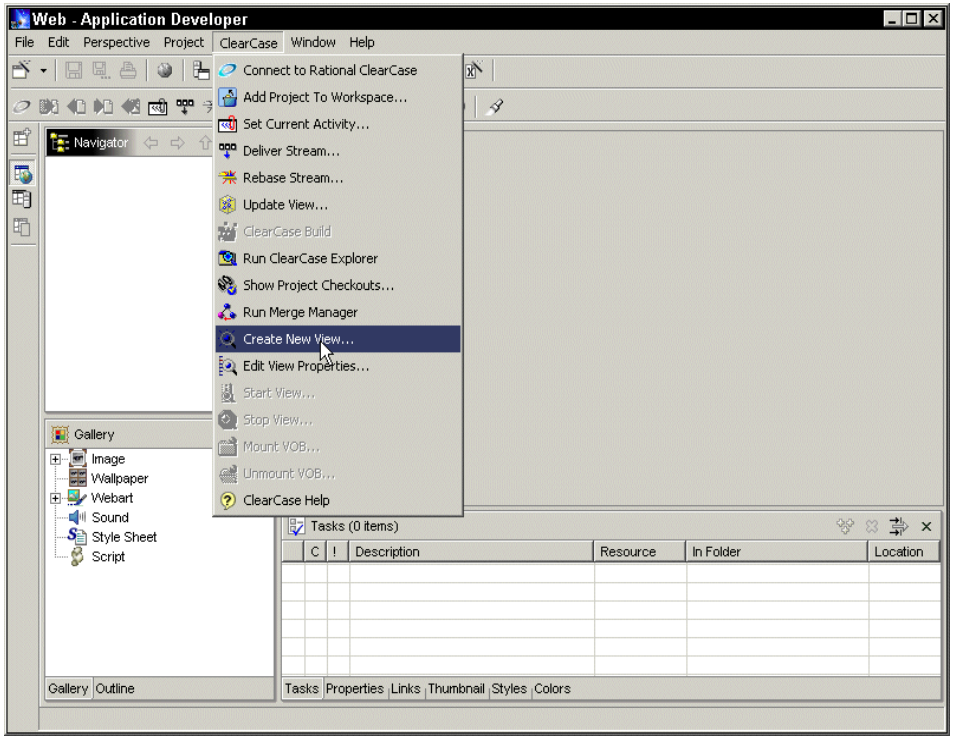


Figure 25-11 Starting the creation of new ClearCase view

The View Creation Wizard opens (Figure 25-12). You use this wizard to set up the view by assigning it a name and other attributes.

In our example we don't want to use ClearCase UCM so you should answer "no" to the question "**Will this view be used for working on one of the Projects described in the ClearCase project tree?**". Click **Next.**

In the Location dialog choose a location for the snapshot view. ClearCase copies versions of elements from VOBs into the directory you specify. In our example, the location is: c:\myUserID_view. Click **Finish**.

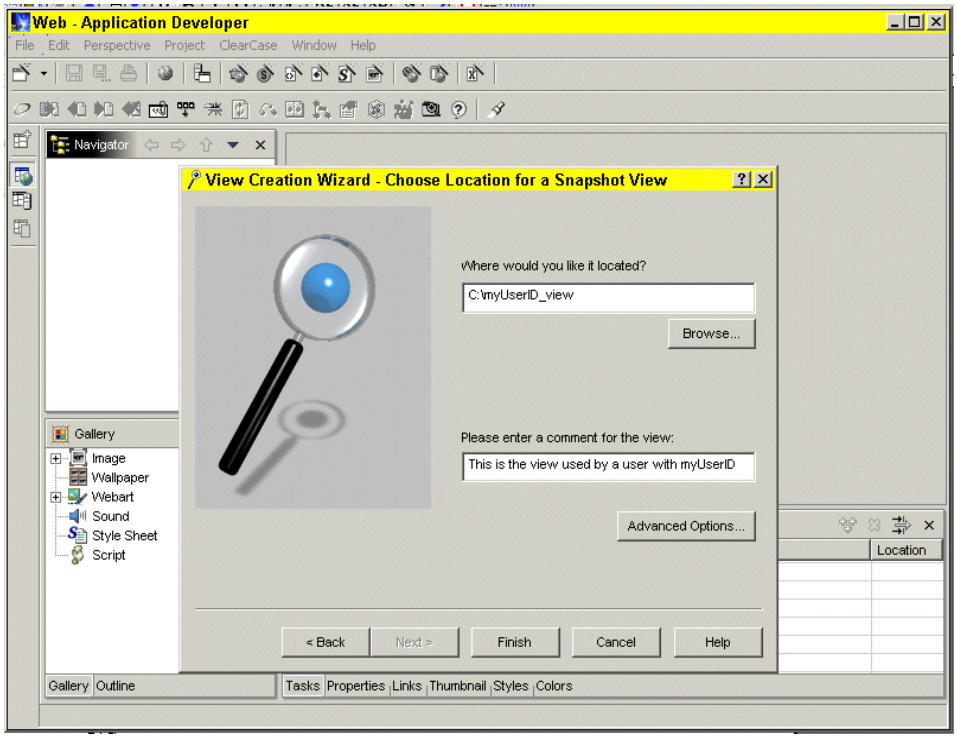


Figure 25-12 Creating new ClearCase view -giving it a name

5. A Confirm dialog opens. Click **OK.** The wizard will now create the view and the VOB namespace browser will be displayed, allowing you to select the content of the view. This dialog is used to select the elements, that is files and directories under ClearCase control, that you want to load into your view or unload from the view.

   If you load a directory element, all of the file and directory elements below the directory will also be loaded into the view. Select the VOB folder that you have created (`WSAD_VOB1`), click **Add**, and then click **OK** (Figure 25-13).

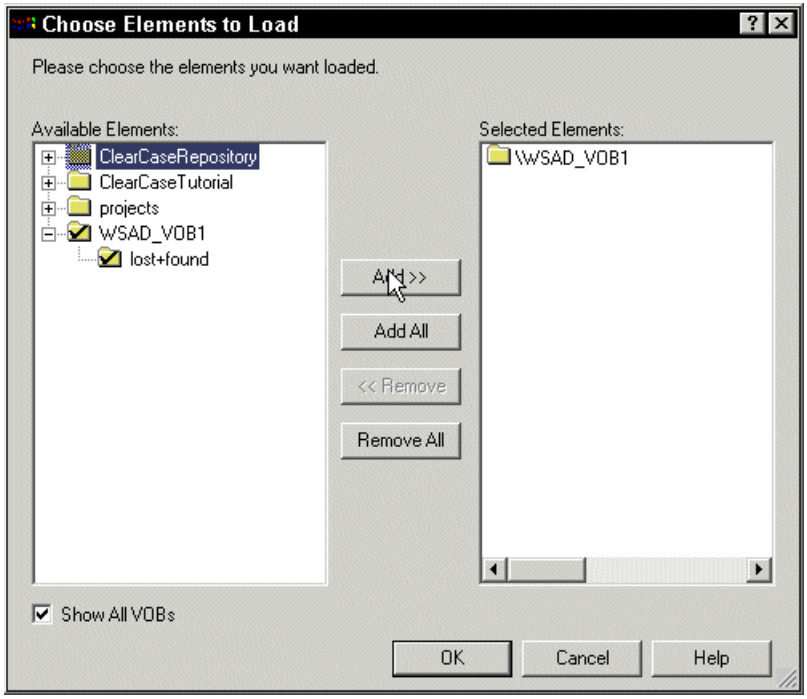


Figure 25-13 Select the VOB to load

6. An an information dialog informs you about the activity being performed. Click **OK** again. Figure 25-14 shows the directory structure that is created.

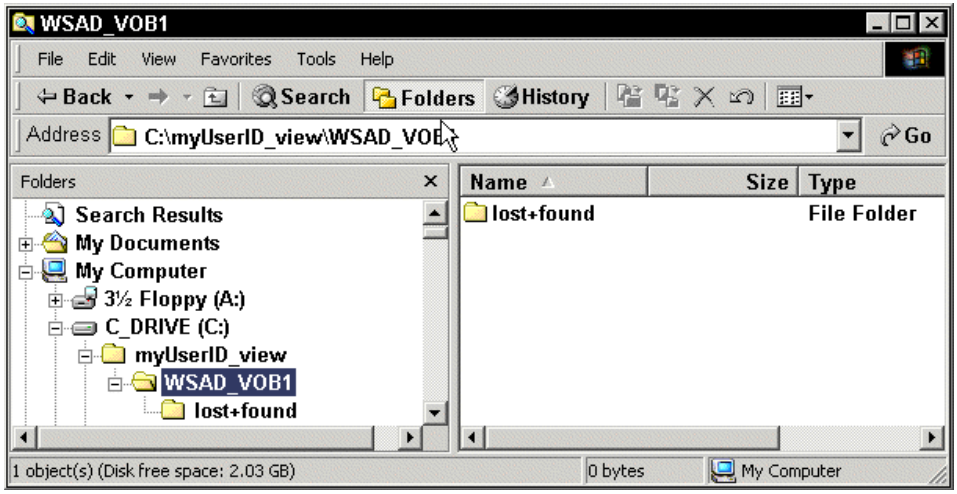


Figure 25-14 The new directory created

7. You are now ready to start working with ClearCase LT.

In the next section we will look at some of the options that are now available.

25.5.3 Move a project into ClearCase

You are now ready to work with ClearCase LT. However the projects in your workspace are not under ClearCase control unless you explicitly specify them to be so.

You can manually move elements that are under ClearCase source control or you can run a script that automates the process. To move files manually, select the project in a Application Developer Navigator View and select **Move Project Into ClearCase** from its context menu (Figure 25-15).

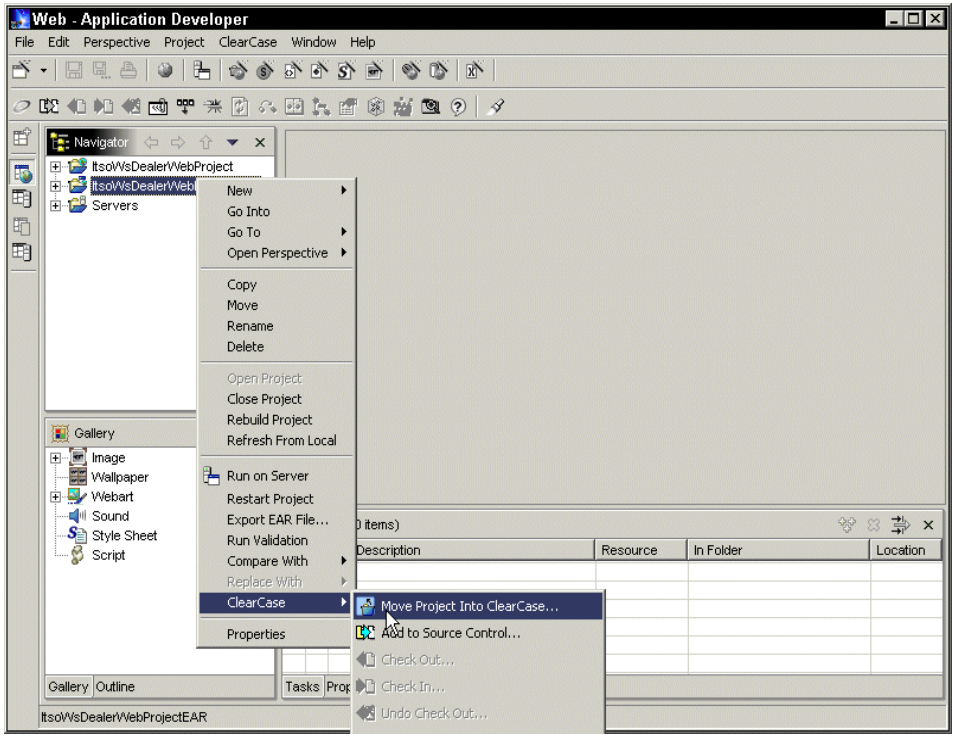


Figure 25-15 Moving project to ClearCase

ClearCase prompts you for the directory path of the VOB. Select the one that was created earlier (`myUserID_view\VOB WSAD_VOB1`) and click **OK** to confirm (Figure 25-16).

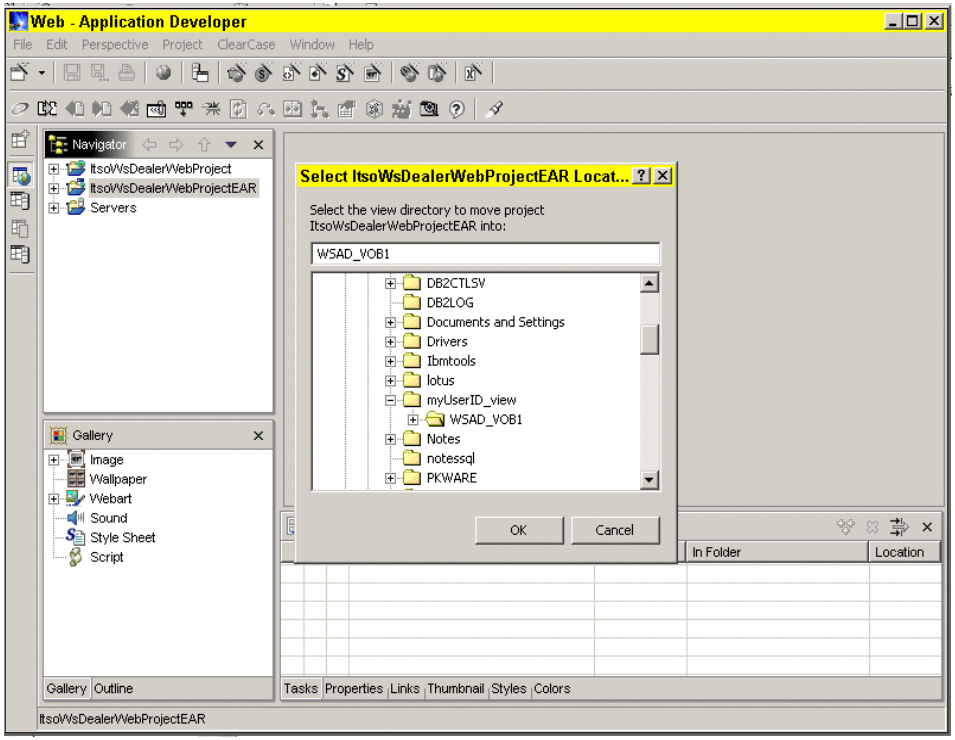


Figure 25-16 Selecting the view for the project (where it will be moved to)

A dialog will be displayed that allows you to select the elements (files) you want to be under ClearCase control (Figure 25-17).

By selecting or deselecting the **Keep checked out** check box you decide whether you want to check the elements in as you place them under ClearCase control, or if you want to leave them checked out to you for further processing.

Confirm your selections by clicking **OK**.

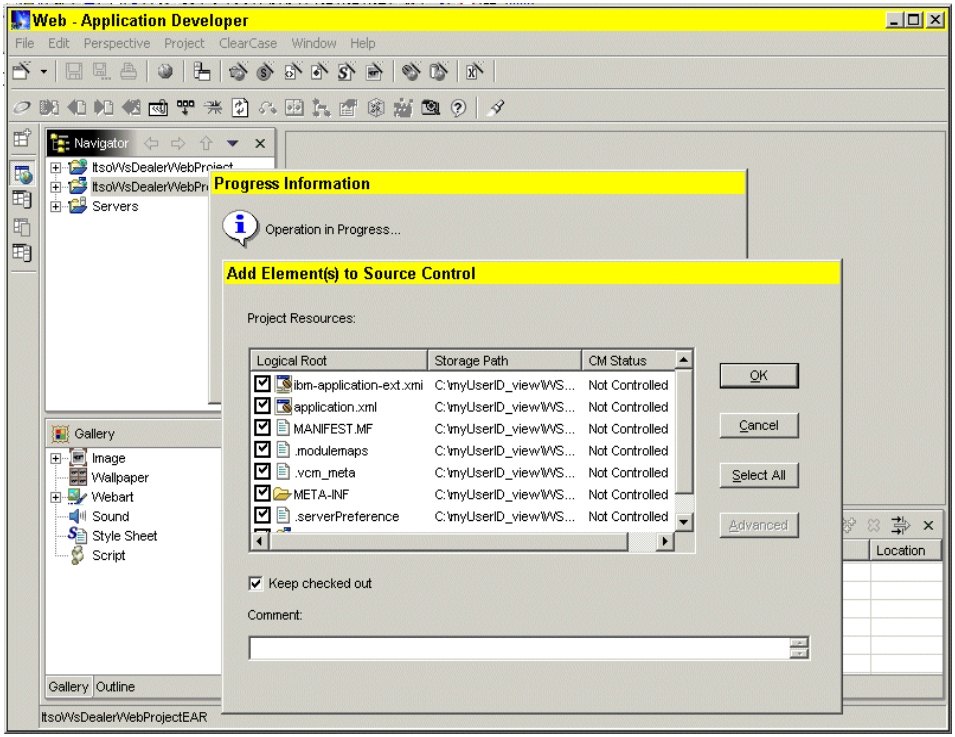


Figure 25-17 Selecting individual files to control

A progress dialog opens showing the task being performed.

When the task is completed a green icon is shown beside each element you have selected. This icon indicates that the element is currently checked out of ClearCase. Also, a blue icon background indicates that the element is now under ClearCase source control (Figure 25-18).

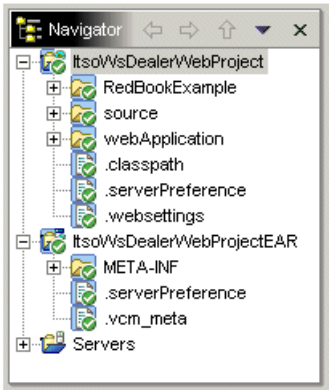


Figure 25-18 Resources controlled by ClearCase marked with special icons

Attention: Note that the elements have been *physically* moved from the Application Developer workspace to the ClearCase view (Figure 25-19).

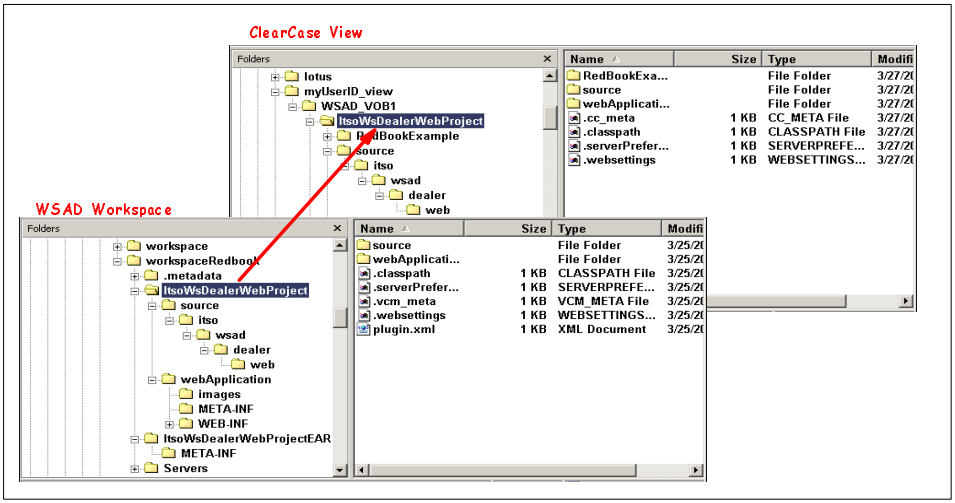


Figure 25-19 Elements moved from the workspace to ClearCase view

If you have many projects to move, it is a good idea to use scripts. Refer to the Application Developer ClearCase help for details on how to do this.

Important recommendation: Do not move projects that you have created and configured for testing in the WebSphere Test Environment to ClearCase control.

Remember that moving projects into ClearCase literally moves the Application Developer projects into another directory. This will invalidate any absolute paths hard coded in the project metadata files. An example is the server-cfg.xml files of the project where you define the WebSphere Server instances and configurations.

To avoid this problem you can create your Server Instances and Configurations only after you have moved the projects under ClearCase control.

25.5.4 Check in a resource

If you left the **Keep checked out** checkbox selected when moving your project under ClearCase control, you will need to check the resources back in once you are finished with them. Let's see how to do this.

To check in a resource that you have previously checked out, select the resource within Application Developer (Navigator view) and select **ClearCase—>Check In** from its context menu or click the **ClearCase Check In** icon  .

A dialog will prompt you to select all resources that you want to be checked in. Make your selections and confirm them by clicking **OK** (Figure 25-20).

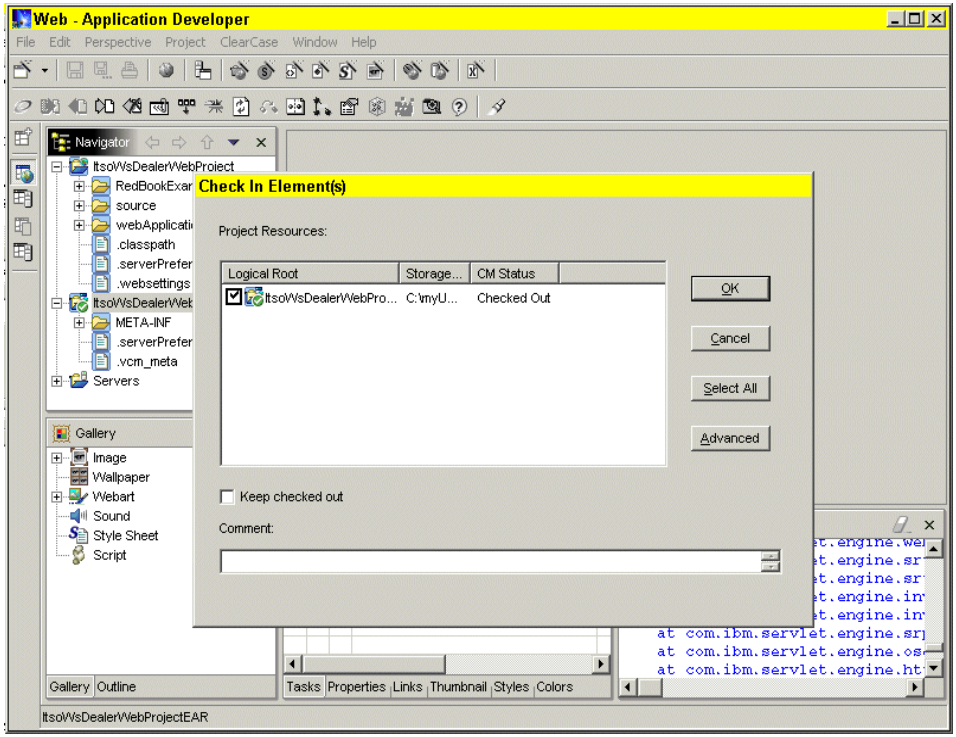


Figure 25-20 Resources to be checked in selection dialog

ClearCase performs the check in action and the checked out icon will be removed for these resources.

25.5.5 Check out a resource

To modify an element that is under ClearCase control, you first need to check it out.

To check out a resource, select it from the Application Developer Navigator view and select **ClearCase—>Check Out** from its context menu. As an alternative click the **ClearCase Check Out** icon (Figure 25-21).

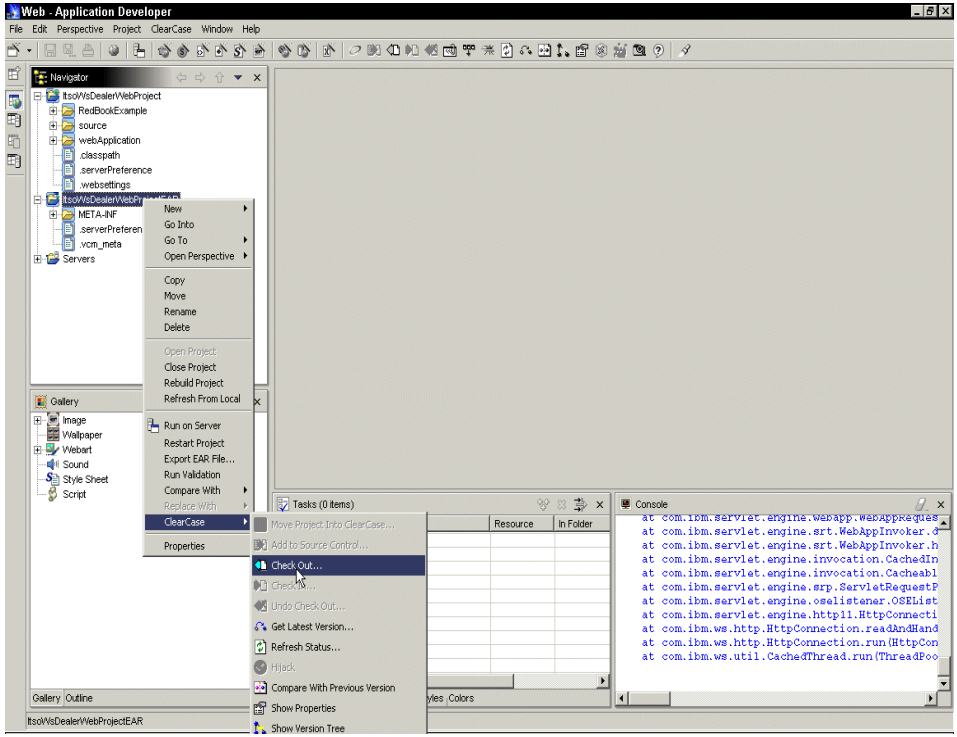


Figure 25-21 Resource check out in ClearCase

A dialog will be displayed where you can select from a list of resources that are available for check out (Figure 25-22).

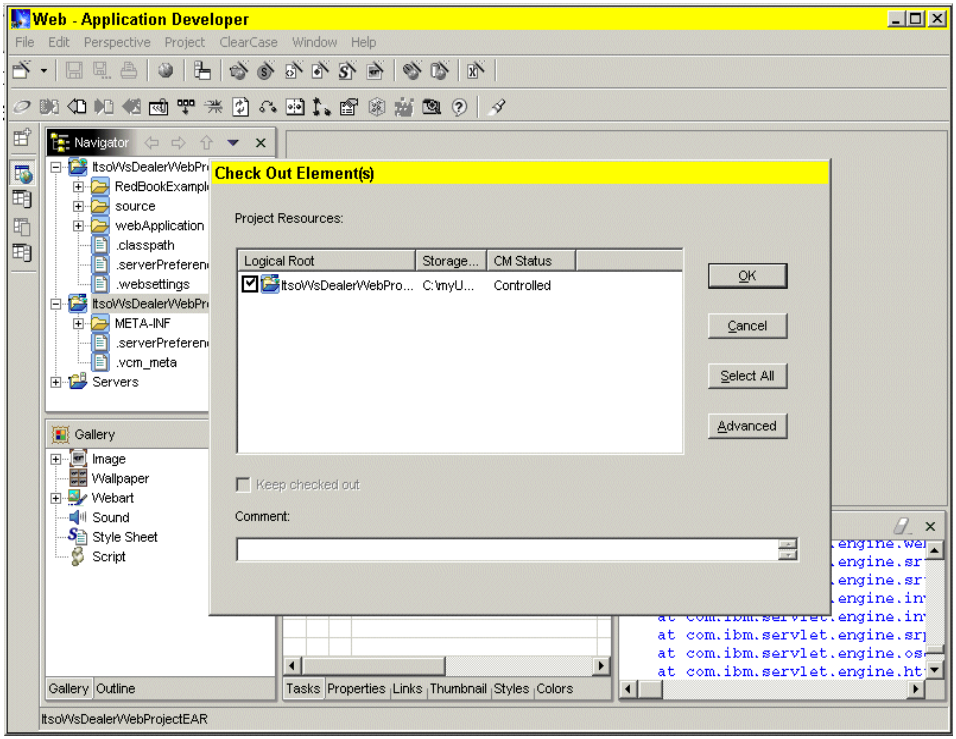


Figure 25-22 Resources to be checked out selection dialog

After you make your selection and confirm it by clicking **OK**, ClearCase performs the check out action and the checked out resources will be marked with the checked out icon  in the Navigator view.

25.5.6 The ClearCase Explorer

The ClearCase Explorer is the native ClearCase user interface. It can be started from Application Developer by selecting **ClearCase—>Run ClearCase Explorer** (Figure 25-23).

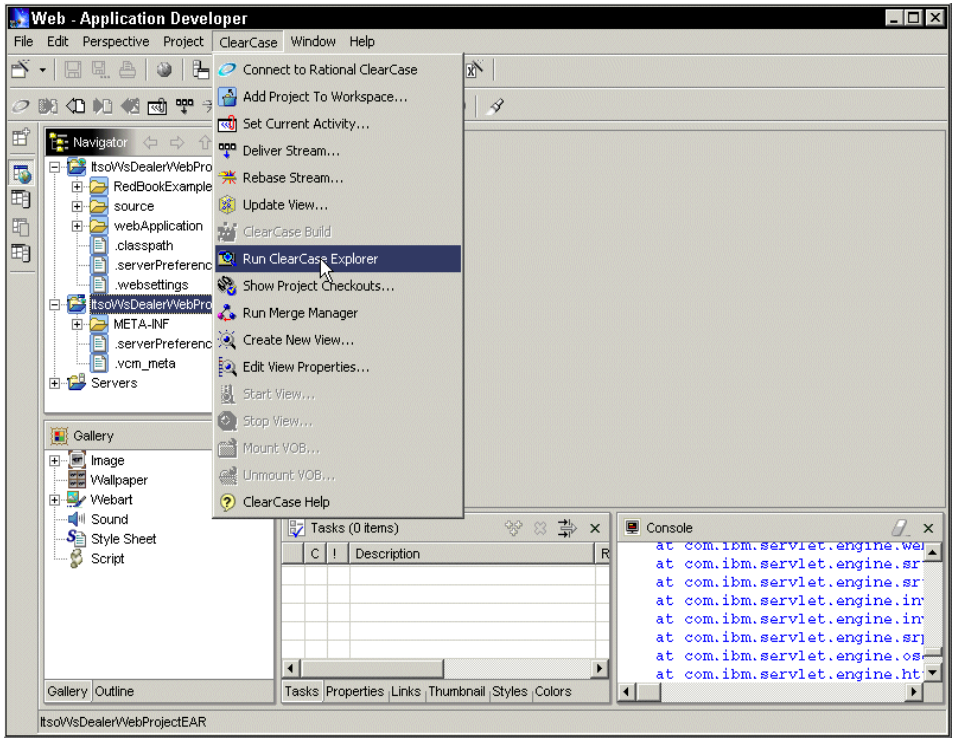


Figure 25-23 Starting the ClearCase explorer

The Clear Case explorer, Figure 25-24, enables you to access activities, source files and directories under ClearCase control, and to issue ClearCase commands.For example, you can complete the following tasks from ClearCase Explorer:

- Join a project (UCM only) and create views.
- Create, find, and set activities (UCM only).
- Check out and check in source files and add new files to source control.
- Deliver activities and rebase your development stream (UCM only).
- Merge versions.
- Update views, remove views, and perform other tasks to maintain your work areas.

You can customize the ClearCase Explorer interface by doing any of the following:

- Adding or removing shortcuts to views, applications, or URLs in the Shortcut Pane.

- Changing the information that ClearCase Explorer displays.
- Adding a command to open a Windows application (such as your favorite editor) by right-clicking a ClearCase object in ClearCase Explorer or Windows Explorer.

You can also change some default ClearCase behaviors from the ClearCase explorer.

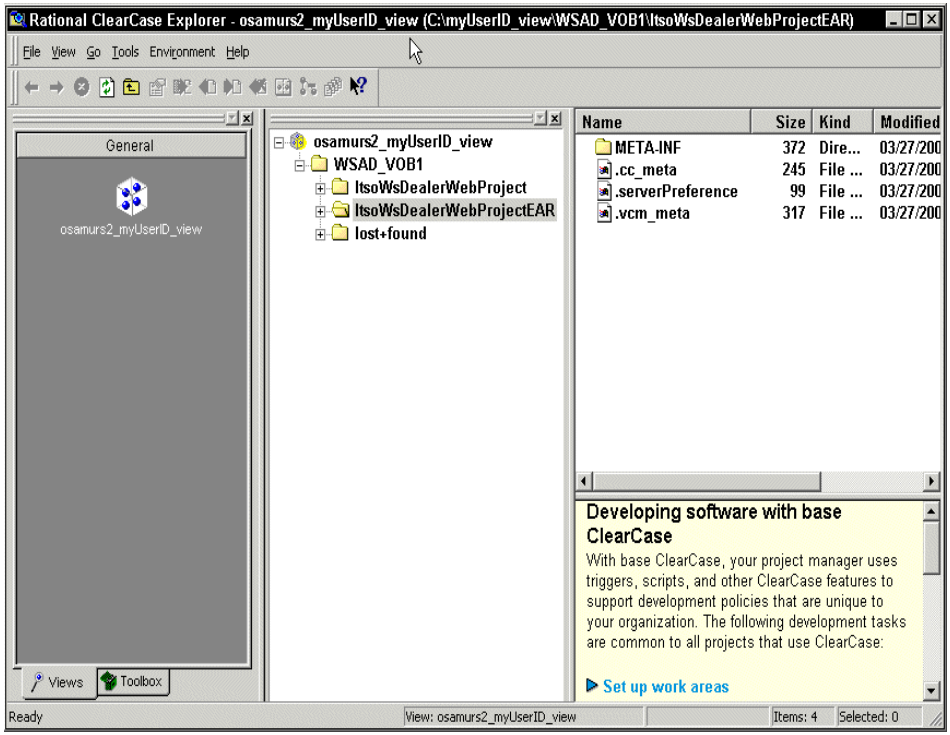


Figure 25-24 ClearCase Explorer

For more information about this tool please see the help for ClearCase in Application Developer.

25.5.7 Conclusion

From within Application Developer you can perform many Rational ClearCase functions. As we have seen, you can execute some tasks from the Rational ClearCase menu. You can also select a file and use the **ClearCase** option from its context menu, or click an icon on the ClearCase toolbar. Table 25-3 is an overview of the Rational ClearCase functions that can be performed from within Application Developer.

Table 25-3 ClearCase functions in Application Developer - overview

ClearCase context menu	ClearCase Menubar menu	ClearCase Toolbar icons
Move Project into ClearCase	Connect to Rational ClearCase	Connect to Rational ClearCase
Add to Source Control	Add Project to Workspace	Add to Source Control
Check Out	Set Current Activity	Check Out
Check In	Deliver Stream	Check In
Undo Check Out	Rebase Stream	Undo Check Out
Get Latest Version	Update View	Set Current Activity
Refresh Status	ClearCase Build (not in LT)	Deliver Stream
Hijack	Run ClearCase Explorer	Rebase Stream
Compare With Previous Version	Show Project Checkouts	Refresh Status
Show Version Tree	Run Merge Manager	Get Latest Version
Show Properties	Create New View	Show Properties
Show History	Edit View Properties	Update View
—	Start/Stop View	ClearCase Build (not in LT)
—	Mount/Unmount VOB	Run ClearCase Explorer
—	ClearCase Help	ClearCase Help

The rest of ClearCase functionality can be accessed from the Rational ClearCase Explorer that can be launched from within Application Developer.

Part 8

The plug-in development environment

Part 8 introduces the plug-in development environment (PDE).

© Copyright IBM Corp. 2002

26

Understanding the PDE

The plug-in development environment (PDE) is a tool that is designed to help you develop plug-ins while working inside the workbench.

This chapter describes:

- Introducing PDE
- Configuring PDE
- Setting up the workbench
- Extension points
- Platform subsystems

© Copyright IBM Corp. 2002

26.1 Introducing PDE

The PDE provides a set of extension contributions (views, editors, perspectives, etc.) that collectively streamline the process of developing plug-ins inside the workbench. The PDE is a perspective of Application Developer and not a separate tool. It blends with the workbench and offers its capabilities through the new perspective.

26.1.1 Platform subsystems

PDE is based on the platform and the Java development tooling (JDT). The platform subsystems, Figure 26-1, typically add visible features to the platform and provide APIs for extending their functionality. Some of these components supply additional class libraries that do not directly relate to an extension point, but can be used to implement extensions. For example, the workbench UI supplies the JFace UI framework and the SWT widget toolkit. The Java development tooling (JDT) implements a full featured Java development environment. JDT allows users to write, compile, test, debug, and edit programs written in the Java programming language. The JDT makes use of many of the platform extension points and frameworks.

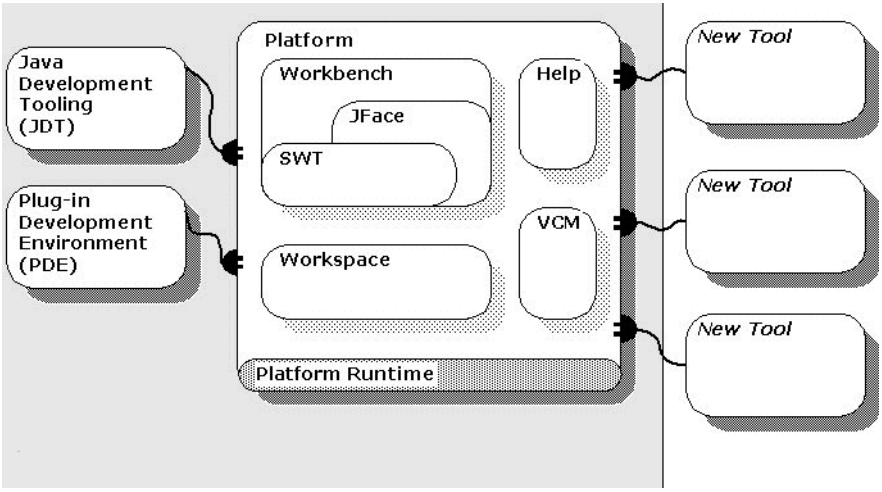


Figure 26-1 Platform, JDT, and PDE

26.1.2 Concepts

PDE manages plug-ins by projects. Each project represents one plug-in and holds the folders and files that define the plug-in and the ways in which it will interact with the platform. These plug-ins will be in your workspace and will show up in the resource navigator or other views that show workspace resources (such as Java Packages).

Host and run-time

One of the most important concepts in PDE is to understand is the concept of host platform instance and run-time platform instance. When you start up the platform, you will use the workbench to work on the projects that define the plug-ins you are building. The workbench instance you are running, as you develop your plug-in using the PDE and other tools, is the host instance. The features available in this instance will come exclusively from the plug-ins that are installed with your platform.

Once you are happy with your plug-in and want to test it, you can launch another platform instance, the run-time instance. This instance will contain the same plug-ins as the host instance, but will also have the plug-ins you were working on in the host instance. PDE launchers will take care of merging your plug-ins with the host plug-ins and creating the run-time instance.

External versus workspace plug-ins

Since the run-time platform instance will represent a collection of features provided by plug-ins from two different places (your current workspace and the original host installation), PDE recognizes these plug-ins as two different "species."

External plug-ins are plug-ins that arrived with the basic platform installation and are simply referenced from their original location without modification. You can reference them, browse them, view their source and debug them, but they are considered read-only.

Workspace plug-ins are those plug-ins under development in your host workbench. They are under your control and can be added, deleted and modified in any way.

External and workspace plug-ins are treated differently by PDE because of their location. For example, opening the plug-in manifest of a workspace plug-in will launch an editor on that manifest file in the workspace. The same operation for an external plug-in will launch an editor on an external URL that references the plug-in's manifest file in the host install location.

26.2 Configuring PDE

Since PDE is a one of the perspectives, you are ready to use with the default setting. However, some setup is offered to create a clean working environment.

26.2.1 Run-time instance configuration

By default, the path for the runtime workspace is **run-time workspace** directory under the **Application Developer** directory used during installation. This is the path of the platform installation used to run and debug your plug-ins. This value is quite acceptable when you want to create plug-ins that will install into a platform configuration that is similar to your host setup. It is important to use different workspaces for your host and run-time platform. Your host workspace location (where all your projects will go on your hard disk) is set when you launch your host development platform. To set up the location of your run-time workspace, select **Window—>Preferences** and choose the "Plug-in Development" page (Figure 26-2). Using this window, you can setup trace options for active plug-ins.

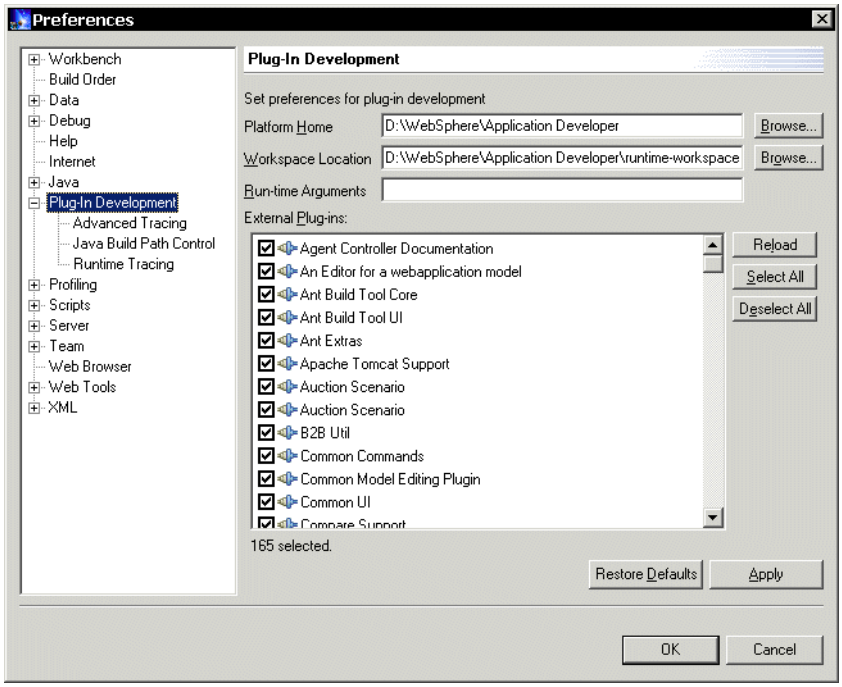


Figure 26-2 PDE preferences

26.2.2 Selecting external plug-ins

The external plug-ins in the preference window is a list of the plug-ins that you want to use in the run-time workspace. If you installed your currently developed plug-in as an external plug-in, and you are going to test your new plugin, two versions of plug-ins will be in conflict with each other. To avoid this conflict, you need to disable an external plug-in from this window before you start the runtime.

Note: This change does not affect to your current host setting.

26.3 Setting up the workbench

To start developing your plug-in, you need to switch to PDE perspective which arranges the views differently and defines placeholders for other views you will likely need (Java Packages and Hierarchy). To change the perspective, use **Perspective—>Open—>Other** and choose **Plug-in Development** from the offered list (Figure 26-3).

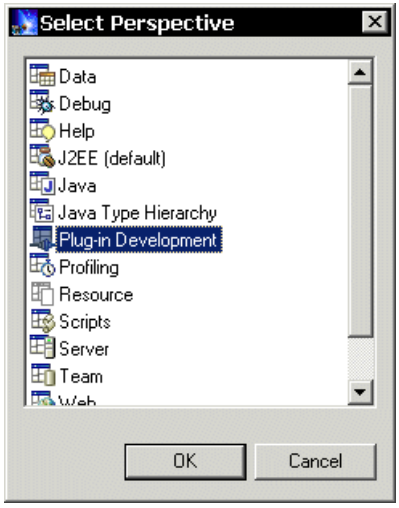


Figure 26-3 Selecting PDE perspective

Figure 26-4 shows the PDE perspective. PDE supplies three different views.

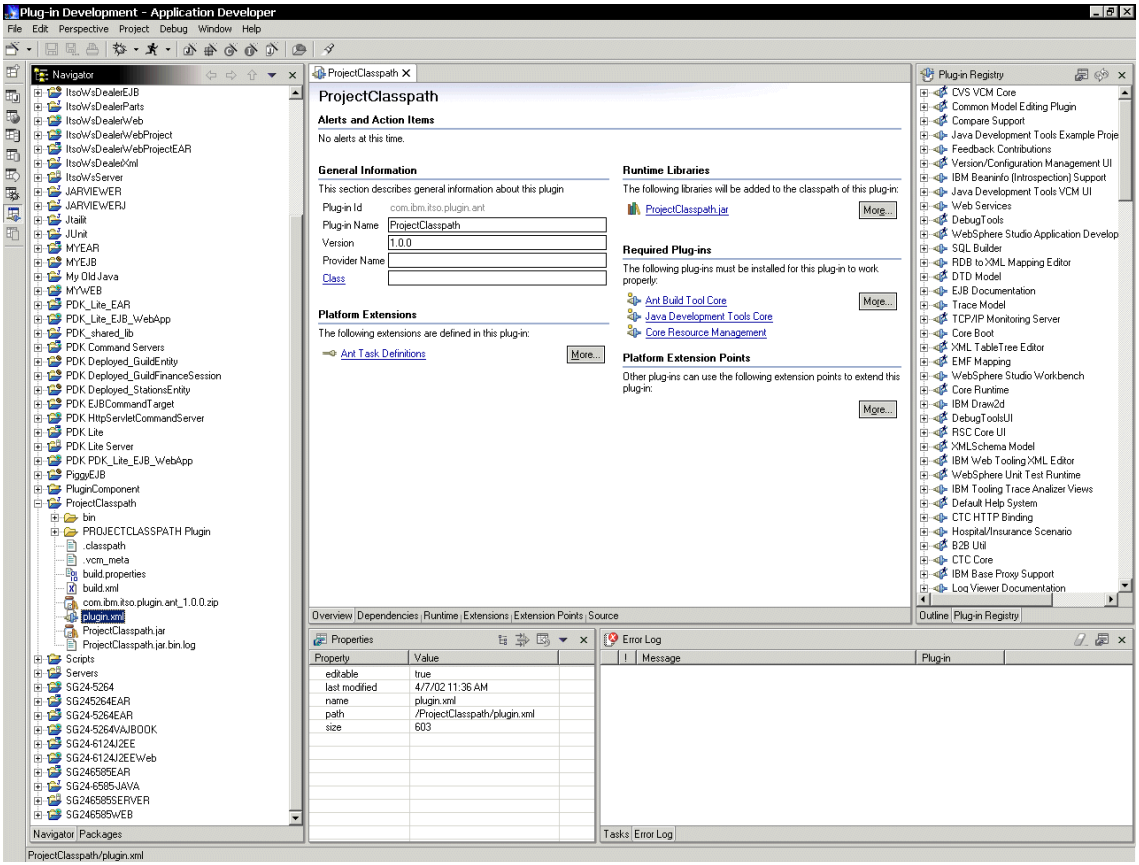


Figure 26-4 PDE perspective

Plug-in manifest editor

The plug-in manifest editor, Figure 26-5, is located in the center by default. The PDE has a number of multi-page editors, including the plug-in manifest editor, that share the same basic behavior. The editor has one source page that shows the raw file content and one or more form pages that present the same information in a more structured format. The plug-in manifest editor partitions the form information into several pages for less clutter. The best way to learn about the plug-in manifest editor is to visit each page.

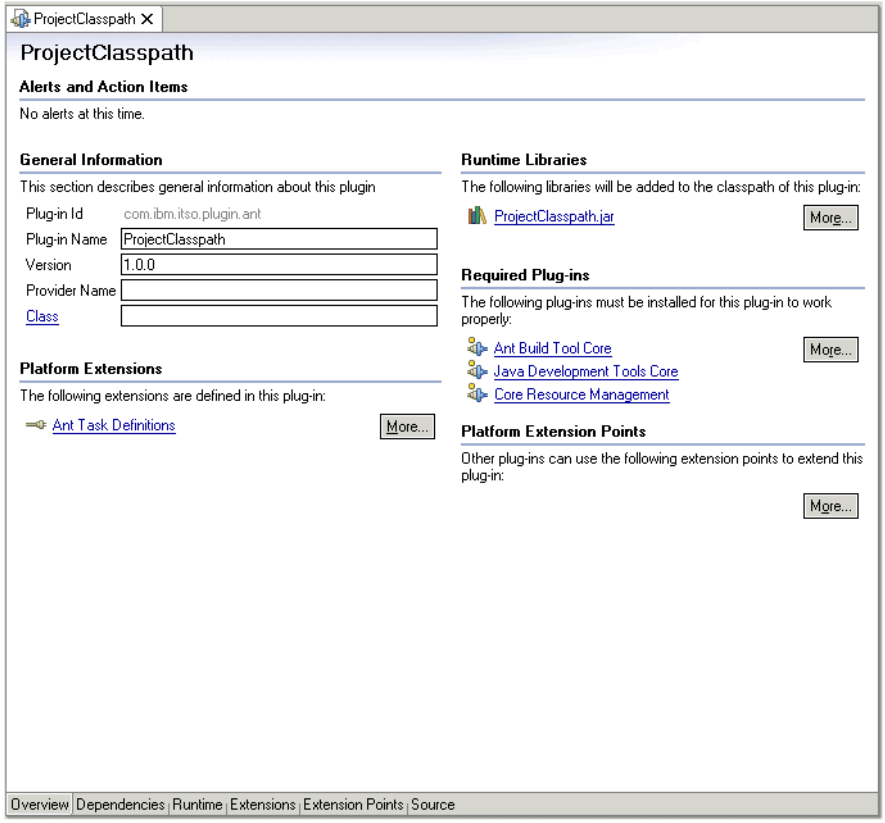


Figure 26-5 Plug-in manifest editor

Error Log

The Error Log view will capture internal errors thrown by the platform or your code. This view is located under the task view by default. This is useful at run-time, since you are likely to cause exceptions during development. As errors are triggered, they will be added to the log view and you can examine details by double-clicking on them. You can also clear the view when you are done with the errors. The same information is stored in the .metadata location of your workspace in the .log file, but the Error Log is much more convenient.

Plug-in Registry

The Plug-in Registry view, Figure 26-6, shows you a read-only view of the plug-in registry of the instance in which it is running. This view can be useful in the run-time platform instance to see whether your plug-in has been started and what plug-in artifacts (extensions, extension points, libraries, etc.) are recognized by the platform. Detailed properties of the selected objects in the Registry view

are shown in the Properties view.The Registry is not shown by default in the PDE perspective. To view the registry, choose **Perspective->ShowView->Other** and choose **Plug-in Registry** from the **PDE Runtime** category in the Show View dialog.

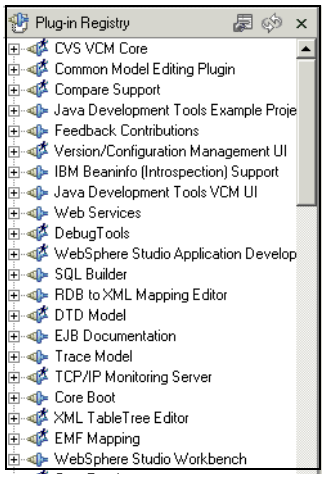


Figure 26-6 Plug-in registry

26.4 Extension points

Extensions are the key mechanism that a plug-in uses to add new features to the platform. Extensions cannot be arbitrarily created. They are declared using a clear specification defined by an extension point. The Eclipse platform is structured around the concept of extension points. Extension points are well-defined places in the system where other tools (called plug-ins) can contribute functionality. Each extension must conform to the specification of the extension point that it is extending. Each extension point defines attributes and expected values that must be declared by an extension. This information is maintained in the platform plug-in registry. Extension point providers query these values from the registry, so it's important to ensure that your plug-in has provided the expected information.

In the most rudimentary form, an extension point declaration is very simple. It defines the ID and name of the extension point. Any other information expected by the extension point is specific to that extension point and is documented elsewhere.

Following list is the platform extension point definitions. To connect your plug-ins with Application Developer, you need to use these extension points. You can learn the details by using the reference document in the help.

- Platform runtime
  - org.eclipse.core.runtime.applications
  - org.eclipse.core.runtime.urlHandlers
- Workspace
  - org.eclipse.core.resources.builders
  - org.eclipse.core.resources.markers
  - org.eclipse.core.resources.natures
- Workbench
  - org.eclipse.ui.actionSets
  - org.eclipse.ui.dropActions
  - org.eclipse.ui.editors
  - org.eclipse.ui.editorActions
  - org.eclipse.ui.elementFactories
  - org.eclipse.ui.exportWizards
  - org.eclipse.ui.importWizards
  - org.eclipse.ui.newWizards
  - org.eclipse.ui.perspectives
  - org.eclipse.ui.perspectiveExtensions
  - org.eclipse.ui.popupMenus
  - org.eclipse.ui.preferencePages
  - org.eclipse.ui.propertyPages
  - org.eclipse.ui.viewActions
  - org.eclipse.ui.views
- Other
  - org.eclipse.ant.core.antObjects
  - org.eclipse.ant.core.antTasks
  - org.eclipse.ant.core.antTypes
  - org.eclipse.compare.contentMergeViewers
  - org.eclipse.compare.contentViewers
  - org.eclipse.compare.structureCreators
  - org.eclipse.compare.structureMergeViewers
  - org.eclipse.core.target.targets
  - org.eclipse.help.contexts
  - org.eclipse.help.contributions
  - org.eclipse.help.support

The reference documentation is useful, but it does not enable any programmatic help for validating the specification of an extension. For this reason, PDE introduces an extension point schema that describes extension points in a format fit for automated processing.

Your plug-in can be an extension point using Extension schema editor. Extension point schema is a valid XML schema as defined by W3C specification. However, the full XML schema specification is very complex and mostly unnecessary for this particular use. For this reason, PDE uses only a subset of XML schema features. Each extension point schema is a valid XML schema, but PDE does not use all the available features.

26.4.1 Plugging into the workbench

The workbench is just a frame that can present various visual parts. These parts fall into two major categories: views and editors.

- A **view** is typically used to navigate a hierarchy of information, open an editor, or display properties for the active editor. For example, the navigator view allows you to navigate the workspace hierarchy. The properties and outline views show information about an object in the active editor. Any modifications that can be made in a view (such as changing a property value) are saved immediately.
- An **editor** is typically used to edit or browse a document or input object. Modifications made in an editor follow an open-save-close model, much like an external file system editor. The platform text editor and Java editor are good examples of editors.

org.eclipse.ui.views

A view is a workbench part that can navigate a hierarchy of information or display properties for an object. Views are often provided to support a corresponding editor. For example, an outline view shows a structured view of the information in an editor. A properties view shows the properties of an object that is currently being edited.

org.eclipse.ui.viewActions

It is common for plug-ins to contribute behavior to views that already exist in the workbench. This is done through the org.eclipse.ui.viewActions extension point. This extension point allows plug-ins to contribute menu items, submenus and tool bar entries to an existing view's local pull-down menu and local tool bar.

org.eclipse.ui.editors

An editor is a workbench part that allows a user to edit an object (often a file). Editors operate in a similar manner to file system editing tools, except that they are tightly integrated into the platform workbench UI. An editor is always associated with an input object (IEditorInput). You can think of the input object as a document or file that is edited. Changes made in an editor are not committed until the user saves them.

Only one editor can be open for any particular editor input in a workbench page. For example, if the user is editing readme.txt in the workbench, opening it again in the same perspective will activate the same editor. (You can open another editor on the same file from a different workbench window or perspective). Unlike views, however, the same editor type (such as a text editor) may be open many times within one workbench page for different inputs.

org.eclipse.ui.editorActions

The org.eclipse.ui.editorActions extension point allows a plug-in to add to the workbench menus and tool bar when another plug-in's editor becomes active.

org.eclipse.ui.popupMenus

The org.eclipse.ui.popupMenus extension point allows a plug-in to contribute to the popup menus of other views and editors.

You can contribute an action to a specific popup menu by its id (viewerContribution), or you can contribute an action for an object type (objectContribution).

- A viewerContribution will cause the menu item to appear in the popup menu of a view or editor specified by id in the markup.
- An objectContribution will cause the menu item to appear in all popup menus for views or editors that have objects of the specified type selected.

org.eclipse.ui.actionSets

Your plug-in can contribute menus, menu items, and tool bar items to the workbench menus and toolbar using the org.eclipse.ui.actionSets extension point. In order to reduce the clutter of having every plug-in's menu contributions shown at once, the contributions are grouped into action sets which can be made visible by user preference.

26.4.2 Manipulating Java code

Your plug-in can use the JDT API to create classes or interfaces, add methods to existing types, or alter the methods for types.

Java elements and resources

The Java model is the set of classes that model the objects associated with creating, editing, and building a Java program. The Java model classes are defined in the org.eclipse.jdt.core.* packages. These classes implement Java specific behavior for resources and further decompose Java resources into model elements.

Java elements

The package org.eclipse.jdt.core defines the classes that model the elements that compose a Java program. The JDT uses an in-memory object model to represent the structure of a Java program. This model is hierarchical. Elements of a program can be decomposed into child elements.

The following list summarizes the different kinds of Java elements and their descriptions.

- **IJavaModel**: Represents the root Java element, corresponding to the workspace. The parent of all Java projects.
- **IJavaProject**: Represents a Java project in the workspace. (Child of IJavaModel)
- **IPackageFragmentRoot**: Represents a set of package fragments, and maps the fragments to an underlying resource which is either a folder, JAR, or ZIP file. (Child of IJavaProject)
- **IPackageFragment**: Represents the portion of the workspace that corresponds to an entire package, or a portion of the package. (Child of IPackageFragmentRoot)
- **ICompilationUnit**: Represents a Java source (.java) file.
- **IPackageDeclaration**: Represents a package declaration in a compilation unit. (Child of ICompilationUnit)
- **IImportContainer**: Represents the collection of package import declarations in a compilation unit. (Child of ICompilationUnit)
- **IImportDeclaration**: Represents a single package import declaration. (Child of IImportContainer)
- **IType**: Represents either a source type inside a compilation unit, or a binary type inside a class file.
- **IField**: Represents a field inside a type. (Child of IType)
- **IMethod**: Represents a method or constructor inside a type. (Child of IType)
- **IInitializer**: Represents a static or instance initializer inside a type. (Child of IType)

- **IClassFile**: Represents a compiled (binary) type. (Child of IPackageFragment)

27

Adding a plug-in to the workbench

In this chapter, we will demonstrate how to create a plug-in by creating a Jar file content browser. Using this plug-in in Application Developer, will allow the user to see the content of any `jar`, `ear`, `war`, or `zip` files in a view or editor.

© Copyright IBM Corp. 2002

27.1 Creating a project

You start by creating a special plug-in project. PDE provides a wizard for setting up the project. You can access this wizard using **File—>New—>Project** and selecting **Plug-in Project** in the list of available project creation wizards (Figure 27-1).

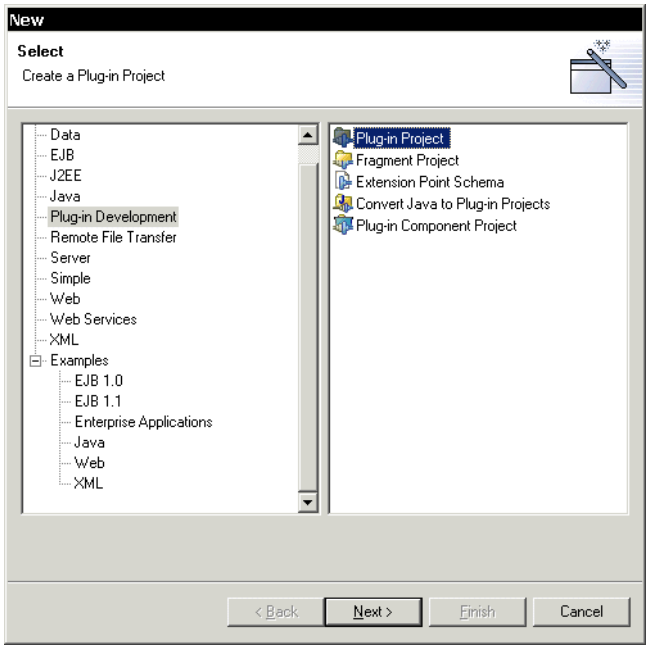


Figure 27-1 Creating a new plug-in project

When you press **Next**, the wizard will be displayed. First you need to specify the project name (Figure 27-2).

Important: Plug-in project names must be unique as they will serve as the unique ID that the plug-in will have.

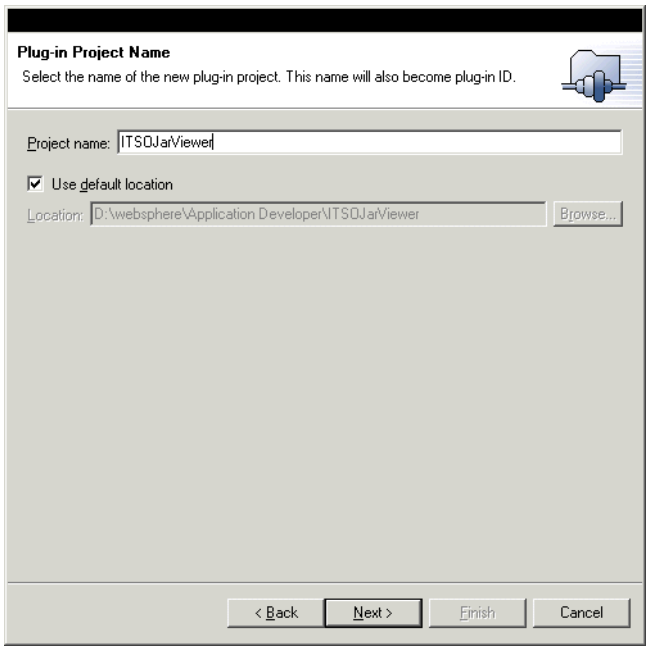


Figure 27-2 Entering the project name

Enter `ITSOJarViewer` as the project name for now and click **Next**.

On the next page you are asked to enter information needed to create the initial structure of the plug-in project (Figure 27-3). In the **Plug-in runtime library** field you need to enter the file that will be used to package the code of your new plug-in, once you are ready to publish it. The **Source folder** will be created to store your Java sources of the new project. For now you can leave the defaults.

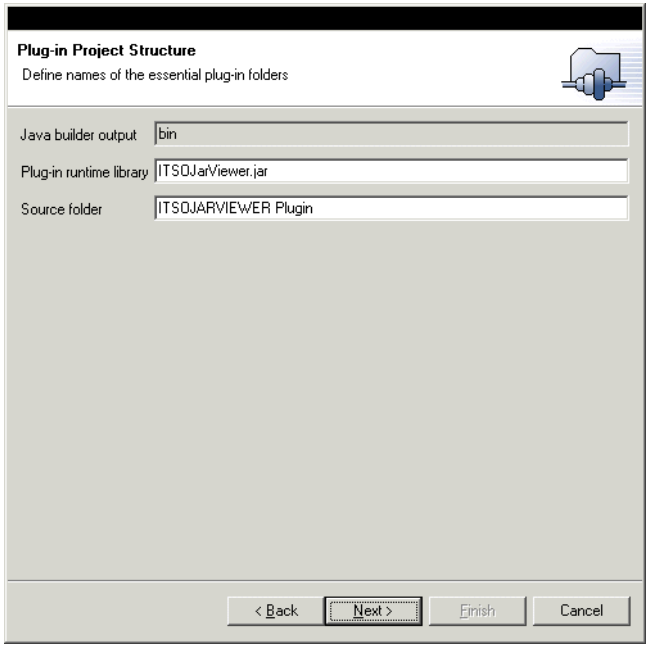


Figure 27-3 Defining the project structure

On the next page, you need to choose what the content of your new plug-in is going to look like. If you choose a *blank plug-in*, only the plug-in project will be created. If you use the *template wizards*, the PDE will also create files such as the plug-in class, build properties, and plug-in manifest. In our example, we leave the defaults and create a blank plug-in (Figure 27-4).

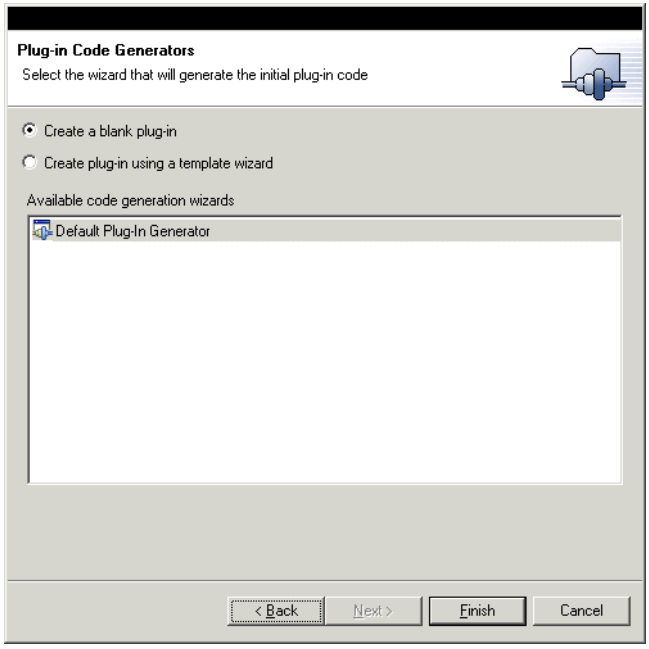

Plug-in Code Generators
Select the wizard that will generate the initial plug-in code
Create a blank plug-in
Create plug-in using a template wizard
Available code generation wizards
Default Plug-In Generator
< Back
Next >
Finish
Cancel

Figure 27-4 Creating a blank plug-in

When you press **Finish**, the wizard will create the new project and put an entry in the Java build path for all the specified folders and files.

Important: The build path is important so that the generated Java classes are correctly built.

After the wizard is finished, the initial project structure should look like Figure 27-5.



Figure 27-5 Project structure

27.2 Creating the plug-in

To create your plug-in, you need to open *plugin.xml*.

The Overview page shows you all the "plug-in at a glance" information grouped into sections. The most important information for each section is shown on the page, with hyperlinks and **More** buttons providing you access to additional information (Figure 27-6).

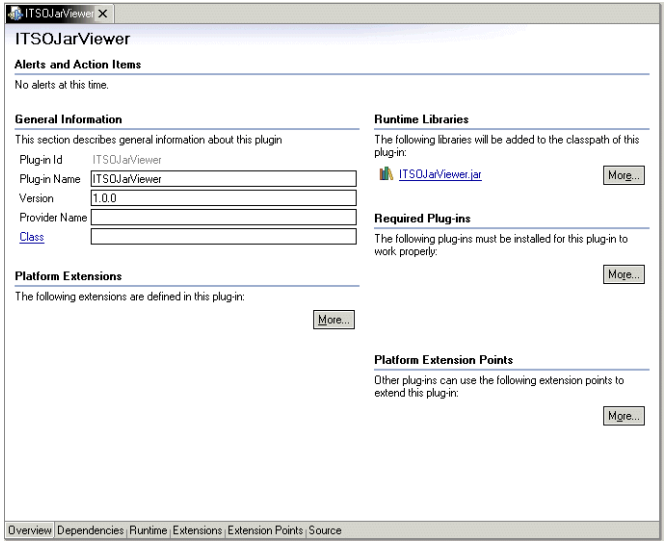


Figure 27-6 Overview page

Go to the Platform Extensions page to create your plug-in. To do so, click the **More** button next to the **Platform Extensions** list (which is now empty).

On the Extensions page click **New** button next to the **All Extensions** area (Figure 27-7).

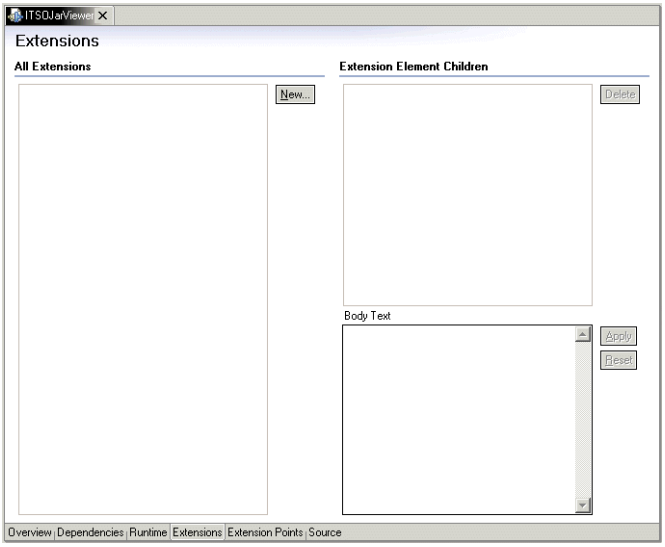


Figure 27-7 New blank extensions page

27.2.1 Selecting an extension point

Since you are going to create a view you need to use the *org.eclipse.ui.views* extension point. Use **Generic Wizards** and **Schema-Based Extensions** then click **Next** (Figure 27-8). All available extension points are listed. Find the Views extension point and confirm that it is org.eclipse.ui.Views.

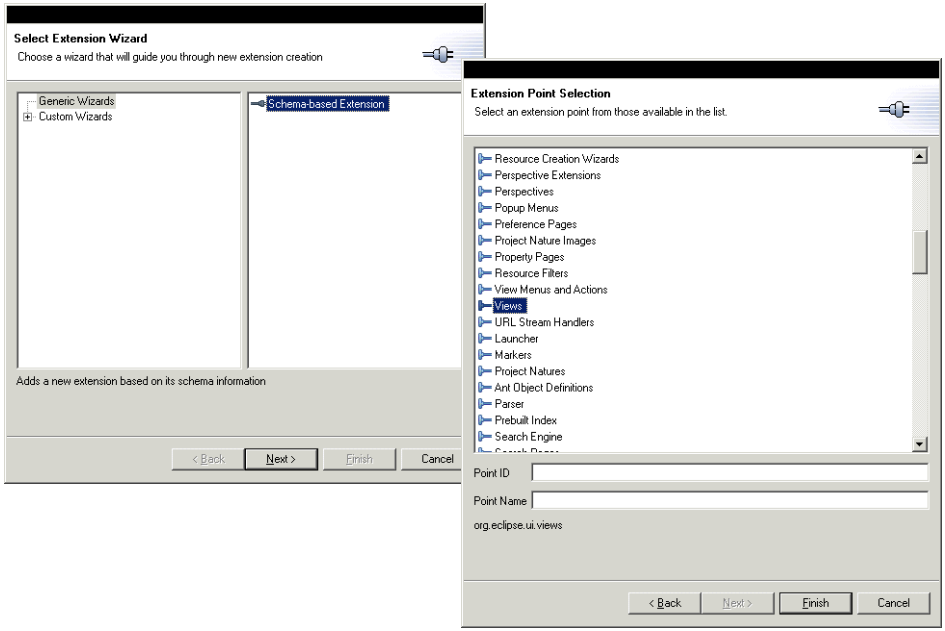


Figure 27-8 Use generic wizards and select views

A warning, Figure 27-9, is displayed informing you that you need to add the required plug-ins in the dependency page. As you are creating your code, you might need to use several existing platform plug-ins. You need to add all of them into the dependency page. Click **OK** to close. Now you can see the Views Extension point in the list and it is pointing to org.eclipse.ui.views in the property.

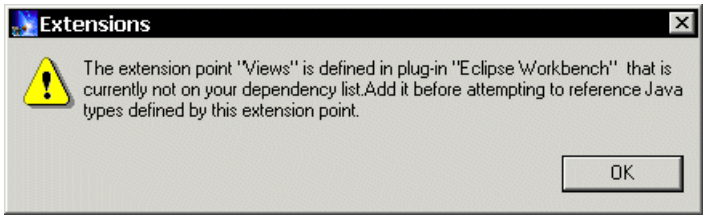

Extensions
The extension point "Views" is defined in plug-in "Eclipse Workbench" that is currently not on your dependency list.Add it before attempting to reference Java types defined by this extension point.
OK

Figure 27-9 Dependency warning

27.2.2 Adding a view

Now you need to add a view. Right click on the Views Extension then select **New—>View** (Figure 27-10). The created view *ITSOJarViewer.view1* represents your new plug-in.

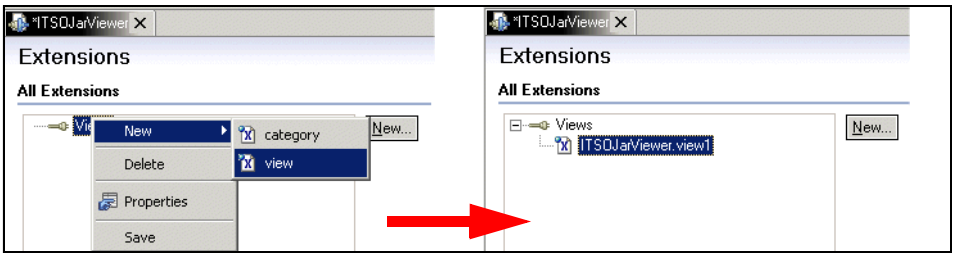


Figure 27-10 Select the view

Now you can go to the property window to create the actual plug-in class.

Note: You may need to add following plug-ins to your Java build path to proceed:

- org.eclipse.ui_1.0.2 (workbench.jar)
- org,eclipse.swt_1.0.1 (swt.jar)

If they are missing, the wizard generates wrong code.

In the Property window, Figure 27-11, click the **class** property once and then click the down-arrow of the displayed combo box.

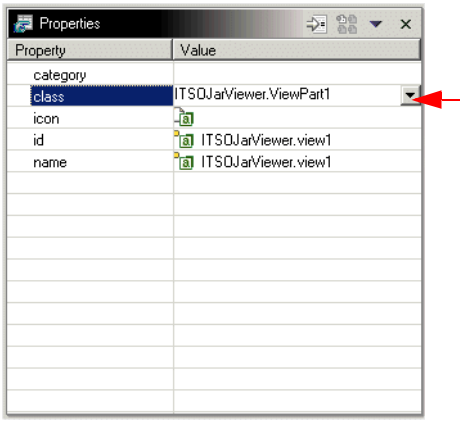


Figure 27-11 Property window

The **Java Attribute Page Wizard,** Figure 27-12, will open and you need to create new plug-in class. Select **Generate new Java class** and change the package and class as follows.

package: com.ibm.itso.plugin
class: ITSOJarViewer

You also can change the ID and name to more readable names as follows.

id: com.ibm.itso.plugin.ITSOJarViewer
name: ITSOJarViewer

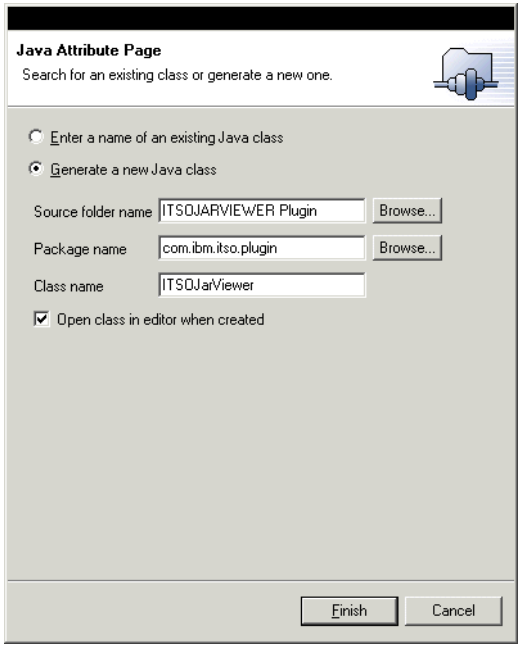


Figure 27-12 Java attribute page

27.2.3 Implementing the ITSOJarView class

Once the class is generated, the source editor will be opened. The plug-in inherits from ViewPart and has two methods and one constructor. You need to implement `createPartControl()` method to create the actual view and setup its content. For this plug-in, you use the JarView class to extract the jar file. This class has a static method taking a jar file name as a parameter and returns all contents' file name, size, and creation date as an array.

```
public static String[][] getListFromJar(String jarname) throws IOException
{
```

We are going to use a table class to show the contents. The table class is in the SWT package and similar to the swing/AWT package. Following code is setting up the table.The table has three columns and each has a header. Each column has a border and resizable.

```
//Setting up the table.
final Table table = new Table(parent,SWT.MULTI);
final TableColumn[] tc = new TableColumn[3];
```

```
for( int i=0; i<3; i++){
   tc[i] = new TableColumn(table,align[i]);
   tc[i].setText(title[i]);
   tc[i].setResizable(true);
   tc[i].setWidth(width[i]);
}
table.setHeaderVisible(true);
table.setLinesVisible(true);
```

The following code will fill up the table with the result from JarView class. TableItem class works as a row set that contains a name, size, and date.

```
private void setupContents(String filename) {
   try{
      String[][] st =
com.ibm.itso.jarview.JarView.getListFromJar(filename);
      for( int i=0; i<(st.length); i++ ){
         TableItem item = new TableItem(table,SWT.DEFAULT);
         item.setText(st[i]);
      }
   }catch(Exception e){
      System.out.println(e);
   }
}
```

The full source code and executable are available in the sample code. See “Locating the Web material” on page 676.

27.2.4 Dependencies

As you need to add a Java build path, you need to specify the dependencies in the plugin.xml. In the dependencies, you need to describe the required plug-ins at *runtime*. In this case, workbench is all you need (Figure 27-13).

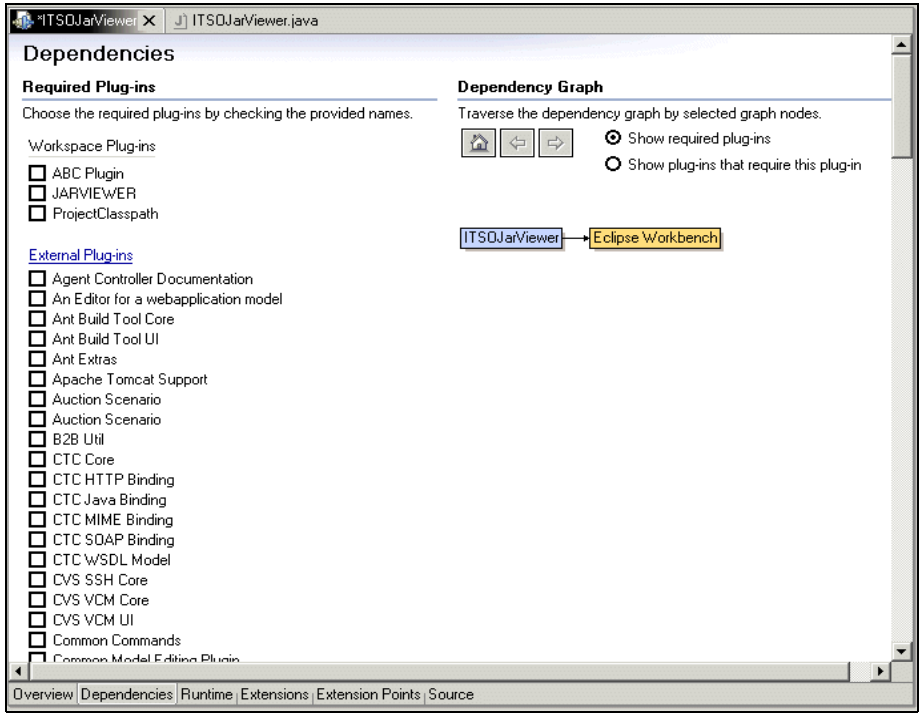


Figure 27-13 Selecting dependencies

27.2.5 Testing your plug-in

Once you have saved everything and do not get any errors, you are ready to run the plug-in.

Select ITSOJarViewer project and click the debug icon in the toolbar. At this time use the *Run-time Workspace*. After a certain time, the new IDE window will show up.

Since you did not create any menu items to invoke this view, the only way to open it is to use the **Perspective** menu. Select **Perspective—>Show view—>Other**. Under the **Other** tree, you can find **ITSOJarView** (Figure 27-14). Select it to display it.

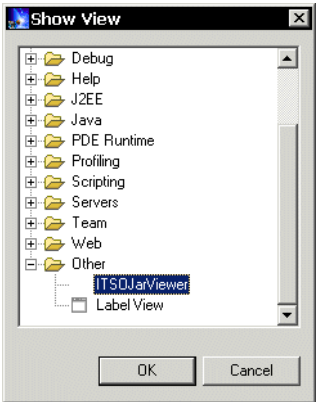


Figure 27-14 Show view

Since you did not pass any jar file to the view, it will try to open `C:/swt.jar` (Figure 27-15).

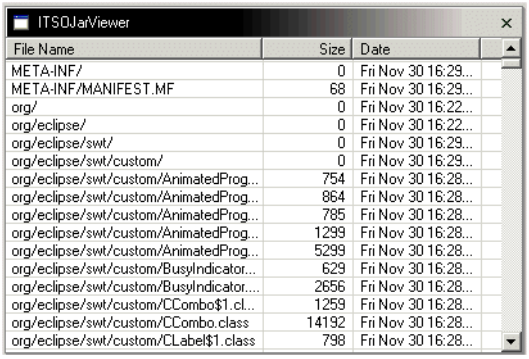

ITSOJarViewer
File Name Size Date
META-INF/ 0 Fri Nov 30 16:29...
META-INF/MANIFEST.MF 68 Fri Nov 30 16:29...
org/ 0 Fri Nov 30 16:22...
org/eclipse/ 0 Fri Nov 30 16:22...
org/eclipse/swt/ 0 Fri Nov 30 16:29...
org/eclipse/swt/custom/ 0 Fri Nov 30 16:29...
org/eclipse/swt/custom/AnimatedProg... 754 Fri Nov 30 16:28...
org/eclipse/swt/custom/AnimatedProg... 864 Fri Nov 30 16:28...
org/eclipse/swt/custom/AnimatedProg... 785 Fri Nov 30 16:28...
org/eclipse/swt/custom/AnimatedProg... 1299 Fri Nov 30 16:28...
org/eclipse/swt/custom/AnimatedProg... 5299 Fri Nov 30 16:28...
org/eclipse/swt/custom/BusyIndicator... 629 Fri Nov 30 16:28...
org/eclipse/swt/custom/BusyIndicator.... 2656 Fri Nov 30 16:28...
org/eclipse/swt/custom/CCombo$1.cl... 1259 Fri Nov 30 16:28...
org/eclipse/swt/custom/CCombo.class 14192 Fri Nov 30 16:28...
org/eclipse/swt/custom/CLabel$1.class 798 Fri Nov 30 16:28...

Figure 27-15 ITSOJarView

27.2.6 Debugging your plug-in

To debug your plug-in, you can add a break point, step execute, inspect the variables, and so on in the usual way from your host workbench (Figure 27-16).

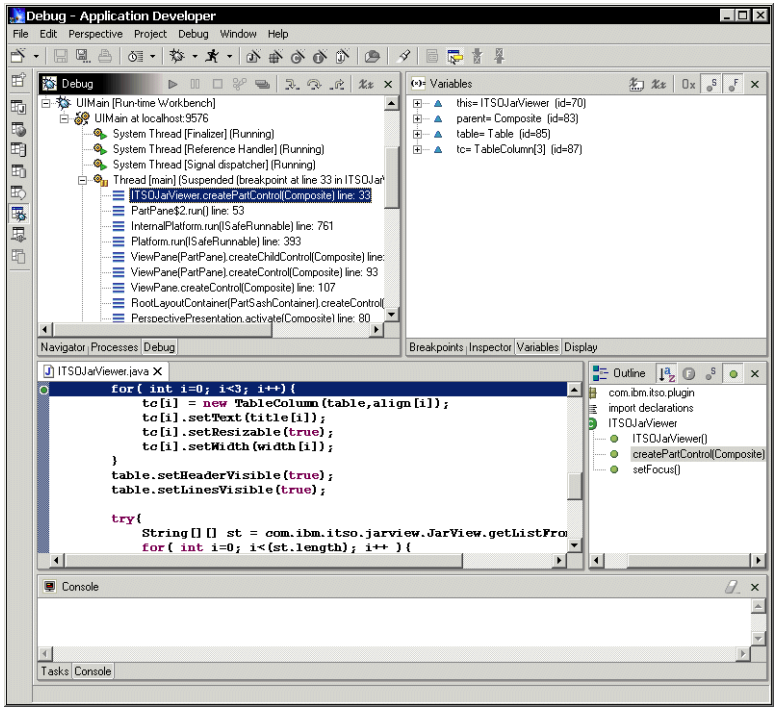


Figure 27-16 Debugging a plug-in

27.2.7 Adding an icon

Notice that since your plug-in does not have an icon yet, you will se an empty icon in the Show View dialog. In the View the default icon is used instead.

To assign an icon to the plug-in you need to specify the icon in the plugin.xml manifest file. We created an icon folder under the project folder and added a jarviewer.gif. So you can now go to the `plugin.xml` and the Extensions view. Select ITSOJarViewer under the viewer extension and edit the icon property to point to this icon (Figure 27-17).

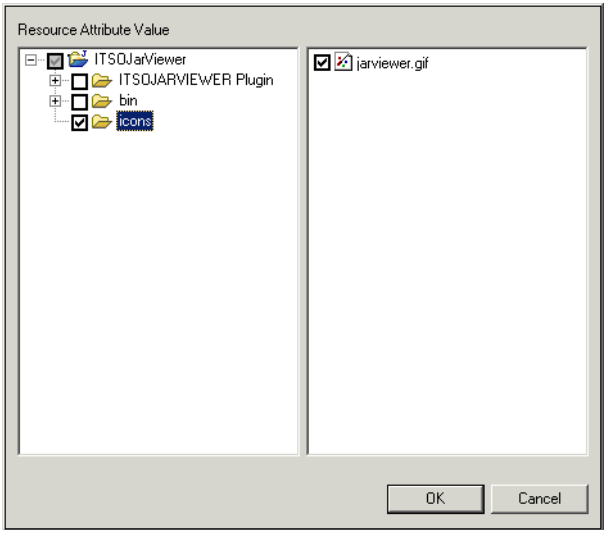


Figure 27-17 Selecting an icon

Once you have selected the icon, save the plugin.xml and re-run. As you can see the icon is used now shown properly as in Figure 27-18.

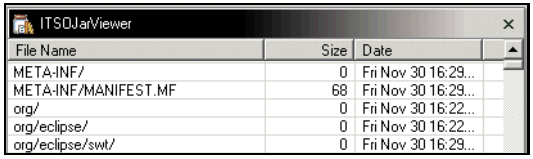


Figure 27-18 Plug-in with icon assigned

27.3 Listening to a selection event

To complete the view, you need to implement a listener to be able to detect which jar file was selected in the Navigator window and should be opened in the viewer. Your class needs to implement the `ISelectionListener` interface and provide the `selectionChanged()` method, to be able listen to the selection from the workbench. If a selection is made in any of views, the workbench will forward the selection event to all interested parts that have registered as a listener.

First of all you need to modify the class definition and add the *implements* section:

```
public class ITSOJarViewer extends ViewPart implements ISelectionListener{
```

Then add the following method call at end of `createPartControl()` method. Now `this` (the viewer class) will become a listener of the workbench page. Every time a selection event occurs, the workbench page will notify your plug-in by sending a message to it.

```
getViewSite().getPage().addSelectionListener(this);
```

When the event is occurred, the `selectionChanged()` method will be invoked. You have to react only on the events which inform you about a file selection. If the object is an instance of IFile, it must be a file. To get full path, you need to call the `getLocation()` method. `setupContents()` method is the method in your plugin to set the file.

```
public void selectionChanged(IWorkbenchPart part, ISelection selection){
   if(selection instanceof IStructuredSelection){
      Object first =
((IStructuredSelection)selection).getFirstElement();
      if(first instanceof IFile){
         setupContents((IFile)first);
      }
   }
}
```

You also need to add org.eclipse.core.resources_1.0.1 (resources.jar) to your build and Core Resource Management to the dependencies in the plugin.xml.

We modified the source slightly as well, to show the file name in the title bar.

```
private void setupContents(IFile file) {

   String filename = file.getLocation().toString();

   if(filename != null){
      setTitle(file.toString());
      table.removeAll();
      try{
         String[][]
st=com.ibm.itso.jarview.JarView.getListFromJar(filename);
         for( int i=0; i<(st.length); i++ ){
            TableItem item = new TableItem(table,SWT.DEFAULT);
            item.setText(st[i]);
         }
      }catch(Exception e){
         System.out.println(e);
      }
   }
}
```

Figure 27-19 shows the completed view. The plug-in receives an event from the Navigation window.

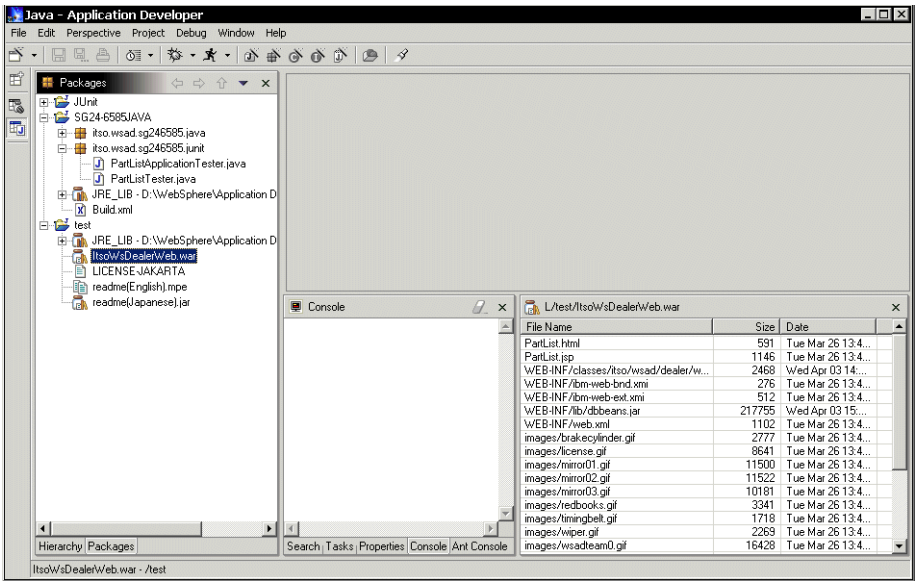


Figure 27-19 JarViewer plug-in

27.4 Creating an editor

The viewer can be used only to view the file contents. To allow the users to modify the file, you need to create an editor. Instead of a view, you can then open multiple editors for each element. In this section we show you how to create an editor using the same class (JarViewer).

You have to create a new ITSOJarEditor project. There are a few differences between view and editor. The most important point is the extension. You need to use the editor extension (org.eclipse.ui.editors) for your new editor (Figure 27-20).

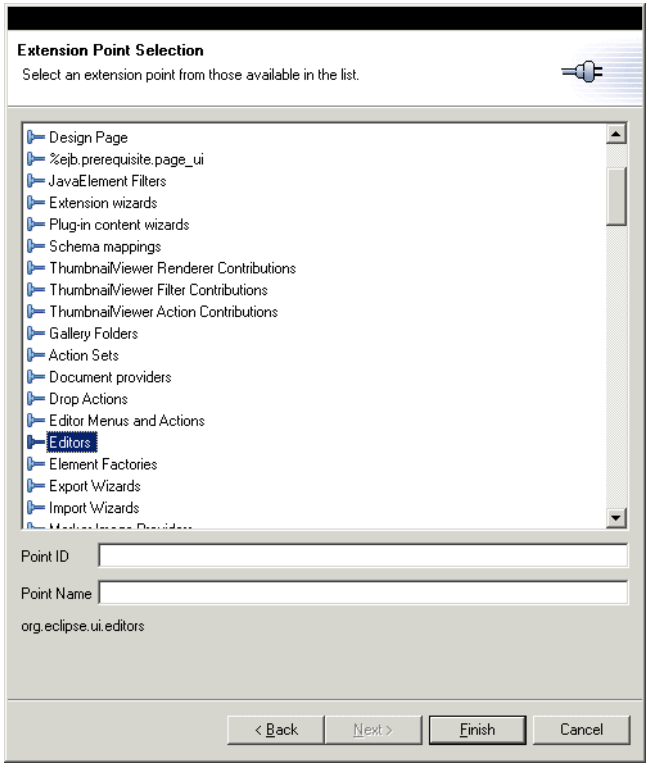


Figure 27-20 Selecting editors extension

Once you created the Editors extension, the all extensions window shows **Internal and External Editor** extension. Use the context menu and select **New**, then **Create new editor.** This process is very similar to creating the view. Edit the properties to fit to your editor and click the **Class** property to create your new editor class (Figure 27-21). Click the **Wizard** button and create your editor class using the Java Attribute wizard (Figure 27-22). The extensions property in the property window is used to specify the file extension that belongs to this editor. It accepts multiple types (comma delimited) and you have to specify `jar, ear, ear, or zip` for this property.

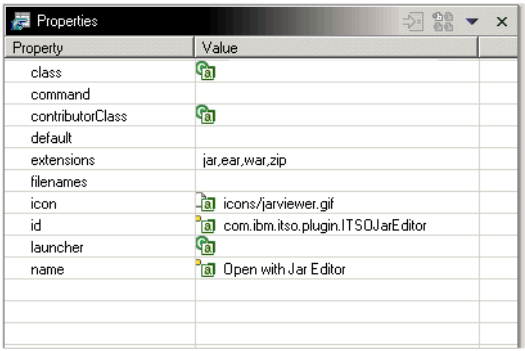


Figure 27-21 ITSOJarEditor properties

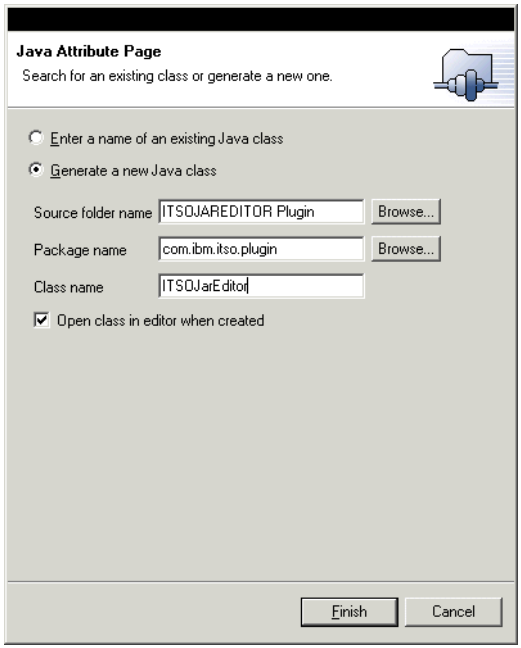


Figure 27-22 Creating an editor class

The Wizard will generate a larger file than it did for the view. It extends EditorPart with some extra methods to implement save/load features. Most importantly is the `init()` method. The `init()` method is invoked when the editor is created. To succeed to instantiate the editor you have to implement the `init()` method.

The init() method

In the `init()` method, you need to set the site and input information using `setSite()` and `setInput()` method. Also you should set the title at this time. Unlike the view, you cannot change the title later.

```
public void init(IEditorSite site, IEditorInput input) throws
PartInitException
{
   if(!(input instanceof IFileEditorInput)){
      throw new PartInitException("Bad request.");
   }
   setSite(site);
   setInput(input);

   IFile file = ((IFileEditorInput)getEditorInput()).getFile();
   filename = file.getLocation().toString();
   setTitle(file.getName());
}
```

The createPartControl() method

On the other hand, the `createPartControl()` method can be similar to the view.

```
public void createPartControl(Composite parent) {
   final Table table = new Table(parent,SWT.MULTI);
   final TableColumn[] tc = new TableColumn[3];

   for( int i=0; i<3; i++){
      tc[i] = new TableColumn(table,align[i]);
      tc[i].setText(title[i]);
      tc[i].setResizable(true);
      tc[i].setWidth(width[i]);
   }
   table.setHeaderVisible(true);
   table.setLinesVisible(true);

   try{
      if(getEditorInput() instanceof IFileEditorInput ) {
         String[][] st =
               com.ibm.itso.jarview.JarView.getListFromJar(filename);
         for( int i=0; i<(st.length); i++ ){
            TableItem item = new TableItem(table,SWT.DEFAULT);
            item.setText(st[i]);
         }
      }
   }catch(Exception e){
      System.out.println(e);
   }
}
```

The other methods

We did not implement the following methods (because we are just displaying the contents and not editing - to make your class a true editor, which edits inside of the jar file, you need to implement them to save or load):

```
//Unimplemented methods.
public boolean isDirty(){ return false; }
public boolean isSaveAsAllowed() {return false;}
public void doSave(IProgressMonitor mon) {}
public void doSaveAs() {}
public void gotoMarker(IMarker marker) {}
```

27.4.1 Running the new plug-in

Once you run the runtime instance, the runtime workbench will appear. Right-click on the file which has jar, war, ear, or zip extension and select **Open With**. You will see your new editor is available.

Figure 27-23 shows the menu and the running result. Three JarEditors are shown as the selection tabs. Notice the below located view. Editors are created in the editors pane and view is created in the viewers pane.

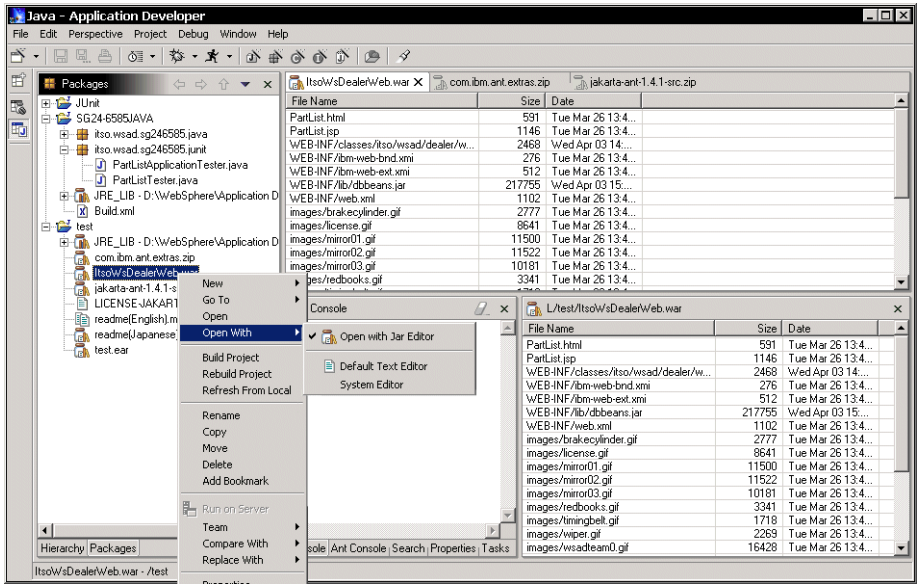


Figure 27-23 Running ITSO Jar Editor

28

Creating AntTask with JDT API

In this chapter, we introduce a plug-in which works as an AntTask.

An AntTask is an extension for the Ant and it allows you to define a subtask for the Ant. We have shown you how to use an AntTask in 20.6, “Working with J2EE” on page 450. Here you can create an AntTask. Because an AntTask is one of the plug-ins, we show you the plug-in and describe how to use JDT APIs.

Note: JDT is a platform subsystem. See 26.1.1, “Platform subsystems” on page 590.

© Copyright IBM Corp. 2002

28.1 Writing a task

As we mentioned, Ant's function is similar to the make tool, but it is platform-independent unlike the make tool. Using XML tags, you can create a variety of build sequences. However you need to setup the class path to the compiler even if you have already set up the Java build path in the workbench.

Ant extra tool is one of the possible solutions, but in this chapter, we introduce a simple class path generator which generates a class path property for the Ant environment using Java build path.

This AntTask plug-in will read your workspace's project information and gather all the project references and external library definitions, as they are described in the `.classpath` file, using JDT API.

The following procedure describes how you can create your own task:

1. Create a Java class that extends org.apache.tools.ant.Task. Use the PDE wizard to do this.
2. For each attribute, write a setter method. The setter method must be a `public void` method that takes a single argument. The name of the method must begin with `set`, followed by the attribute name, with the first character of the name in uppercase, and the rest in lowercase. The type of the attribute can be:
   - String
   - Any primitive type (they are converted for you from their String representation in the beautiful)
   - Boolean - your method will be passed the value true if the value specified in the beautiful is one of true, yes, or on)
   - Class
   - File (in which case the value of the attribute is interpreted relative to the project's basedir)
   - Any other type that has a constructor with a single String argument.
3. If your task has enumerated attributes, you should consider using a subclass of org.apache.tools.ant.types.EnumeratedAttribute as an argument to your setter method.
4. If the task should support character data, write a `public void addText(String)` method.
5. For each nested element, write a create or add method. A create method must be a `public` method that takes no arguments and returns an Object type. The name of the create method must begin with create, followed by the

element name. An add method must be a `public void` method that takes a single argument of an Object type with a no-argument constructor. The name of the add method must begin with `add`, followed by the element name.

6. Write a `public void execute()` method, with no arguments, that throws a BuildException. This method implements the task itself.

28.1.1 Parser time and run time

A task is processed in two phases. One phase is a *parser time* and the other phase is a *run time*. At parser time, is the Ant build initialization processing time, the task is instantiated and initialized. At runtime, the task is executed.

Parser time

- The task gets instantiated using a no-argument constructor.
- The task gets references to its project and location inside the beautiful via its inherited project and location variables.

> **Note:** These are `org.apache.tools.ant.Project` and `org.apache.tools.ant.Location` and not equal to platform plug-in `classes.org.eclipse.core.resources.IProject` or `org.eclipse.jdt.core.IJavaProject`.

- If the user specified an `id` attribute to this task, the project registers a reference to this newly created task.

> **Note:** Tasks can be assigned an ID attribute:
>
> ```
> <taskname id="taskID" ... /
> ```
>
> Where, taskname is the name of the task, and taskID is a unique name for this task. You can refer to the corresponding task object in scripts or other tasks via this name.

- The task gets a reference to the target it belongs to via its inherited target variable.
- The `init()` method is called even if you do not have this, super class Task has this.
- All child elements of the XML element corresponding to this task are created via this task's `createXXX()` methods or instantiated and added to this task via its `addXXX()` methods.

Run time

- All attributes of this task get set via their corresponding `setXXX()` methods.
- The content character data sections inside the XML element corresponding to this task is added to the task via its `addText()` method.
- All attributes of all child elements get set via their corresponding `setXXX()` methods.
- The `execute()` method is called. This is the body of the task. You need to implement your task logic here.

Life-cycle of the task

The above initialization process has been done at parser time, the task has been instantiated and initialized when the build is started. While the initialization steps only occur once, the `execute()`method may be called more than once, if the task is invoked more than once. When the build has done, the task is disposed.

28.1.2 Project class path generator

Since the build cannot get the workspace project's Java build path, we need to add a classpath in the build.xml as follows.

```
<javac srcdir="${src}" destdir="${build}" includes="**/*.java">
   <classpath>
      <fileset dir="/${env.WAS_HOME}/lib"/>
            <include name="/ivjejb35.jar"/>
            <include name="/j2ee.jar">
      </fileset>
   </classpath>
</javac>
```

If you modified your Java build path, you also need to modify this classpath variable. By using our sample, you can reuse the Java build path in your each build task. The following fragment is using our task.

```
<ProjectClasspath pathvar="buildpath"/>
<echo message="build classpath is:${buildpath}"/>
<javac srcdir="${src}" destdir="${build}" includes="**/*.java"
classpath="${buildpath}">
```

ProjectClasspath is the task name and it requires an attribute to pass the Java build path. In this case, we are passing buildpath as the name of path variable. In the javac, classpath will be the value of the buildpath variable. We used echo tag to see the classpath.

28.1.3 Using JDT API

While it is a task plug-in, it seems to be a task for Ant. But to manipulate the project information, we need to use the platform and JDT APIs.

The following is the code of a method that gets an array of the classpath for the specific project name.

```
private IClasspathEntry[] getClasspathEntries(String pjname, boolean resolve)
                                                          throws Exception {
   IProject project =
            ResourcesPlugin.getWorkspace().getRoot().getProject(pjname);
   IJavaProject ijproj = JavaCore.getJavaCore().create(project);
   IClasspathEntry[] cp = ijproj.getResolvedClasspath(resolve);
   return cp;
}
```

IProject belongs to the platform API set and IJavaProject, JavaCore, and IClasspathEntry are belong to JDT API set.

28.2 Creating a task plug-in project

Since a task is a plug-in, you can use PDE to create the task.

Go to PDE perspective and create **ITSOProjectClassPath** project as a plug-in project. Once you created the project, open the plugin.xml and go to the Extensions page, then select a **New generic and schema based extension**. You may see two Ant Task Definitions extensions, because we installed Ant Extra task. Make sure that you choose org.eclipse.ant.core.AntTasks (Figure 28-1).

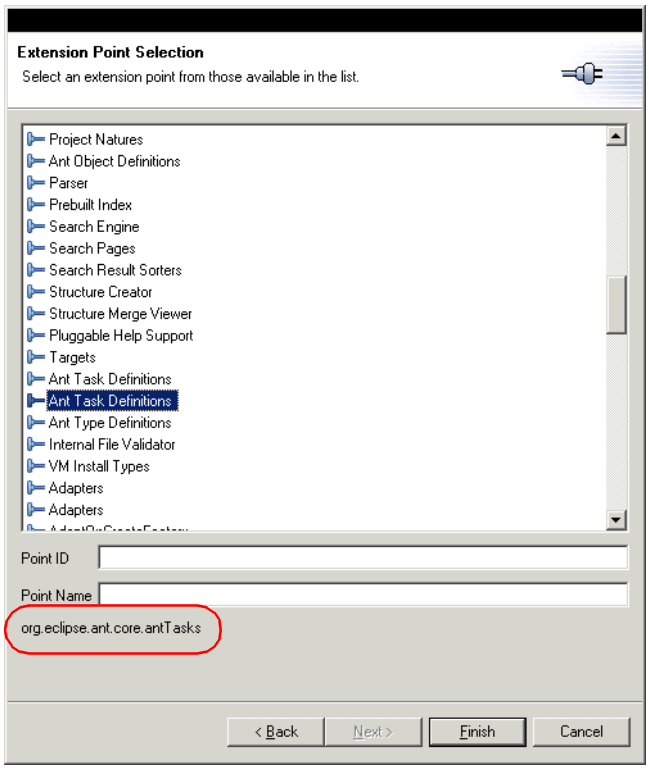


Figure 28-1 Selecting Ant task definition

Click finish and go to the Dependency page and select Ant Task Core as a dependent plug-in. Also we are planning to use platform and JDT API sets, as we mentioned, we suggest you to add **Core Resource Management** and **Java Development Tools Core** plug-in at this time (Figure 28-2).

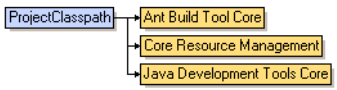


Figure 28-2 Dependencies

Do not forget to change your Java build path. If your new plug-in depends on the other plug-in at run time, such as in this case, you also need that plug-in during the development time. You need to add following libraries.

- plugins/org.eclipse.ant.core/ant.jar
- plugins/org.eclipse.core.resources_1.0.1/resources.jar

- plugins/org.eclipse.jdt.core_1.0.1/jdtcore.jar

28.2.1 Creating a task class

Go back to the Extensions page, and use the context to create **New—>antTask** (Figure 28-3). This is going to be your new task. Edit the name in the property window as `ProjectClasspath`.

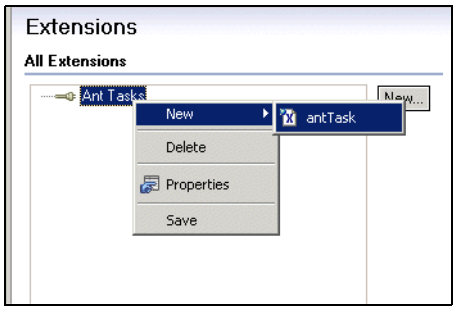


Figure 28-3 Creating an antTask

Now it is time to create your task class. Click the class property and click the button at the right side.

In the Java Attribute Page, change the package as com.ibm.itso.plugin.ant and class as ProjectClasspath, then click **Finish** (Figure 28-4).

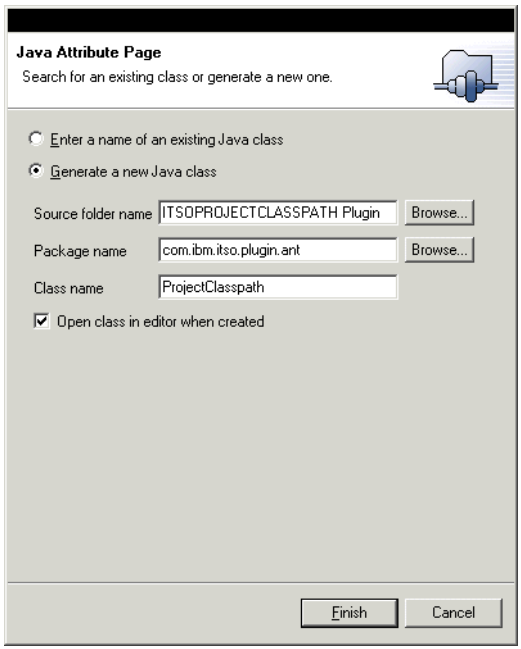


Figure 28-4 Creating a task class

After the code generation has done, the code editor is opened. The task has very simple code as follows.

```
package com.ibm.itso.plugin.ant;

import org.apache.tools.ant.Task;

/**
 * Insert the type's description here.
 * @see Task
 */
public class ProjectClasspath extends Task {
   /**
    * The constructor.
    */
   public ProjectClasspath() {
   }

}
```

Adding a variable

Now you need to have a variable and accessor for it to get/set the **pathvar** variable as follows:

```
private String pathvar;
public void setpathvar(String name){
   pathvar = name;
}
```

Creating the execute method

Now you need to implement the `execute()` method. This must be public void and may throw BuildException.

```
   public void execute() throws BuildException
    {
```

IClasspathEntry is a box to receive the classpath entries from getClasspathEntries method. See 28.1.3, “Using JDT API” on page 629. The variable returnpath is a StringBuffer to keep a “;” delimited classpath string. This will be generated while it runs.

```
   IClasspathEntry[] cp = null;
    StringBuffer returnpath = new StringBuffer();
   try{
```

Then get all classpath entries that belong to this project. As we mentioned, the project name is already passed to this task.

```
          cp = getClasspathEntries(getProject().getName(), true);
```

Now you inspect all entries and take correct path.

```
          for(int i=0; i<cp.length; i++) {

            switch(cp[i].getEntryKind()){
```

Each classpath entry (IClasspathEntry) has an attribute to recognize the characteristic. Variations are SOURCE, LIBRARY,VARIABLE, and PROJECT. The SOURCE is pointing to the source location of this project, so you can ignore this path because javac already know where it is.

```
               case IClasspathEntry.CPE_SOURCE:
                  //this path is for local project source. Nothing to do.
                  break;
```

The LIBRARY and VARIABLE are pointing to the jar, zip, or the other external library. LIBRARY is the item which is attached in the project property. VARIABLE is the item which is attached in the preference window and used in the Java build path as a variable. In both case, you get the path and add to the returnpath.

```
                case IClasspathEntry.CPE_LIBRARY:
                case IClasspathEntry.CPE_VARIABLE:
                   //these paths contain full external path.
                   //just add path to the return string.
                returnpath.append(cp[i].getPath().toString());
                returnpath.append(";");
                break;
```

The last one is PROJECT. It is referencing the other project and it contains only the name of the project. To get the classpath, you need to get the classpath entries that belong to the project and find out the SOURCE at this time. Following getProjectSourceLocation is doing this and returns the path.

```
            case IClasspathEntry.CPE_PROJECT:
               //this path contains only a project name.
               //we need to search full location of the source path element.
               returnpath.append(
                     getProjectSourceLocation(cp[i].getPath().toString()));
               break;

            default:
               break;
          }
         }
         //System.out.println(returnpath);

      }catch(Exception e){
          throw new BuildException(e);
      }
```

Finally, when the classpath has been generated, save it to the specified variable.

```
      this.getProject().setUserProperty(pathvar, returnpath.toString());
   }
```

28.2.2 Running a task

To test the task, select the project and click the debugger button to start run-time workspace. We used a project which is referencing to JUnit project.

Note: All reference project must be in the run-time workspace.

Select **run Ant** menu on the `build.xml` file (Figure 28-5).

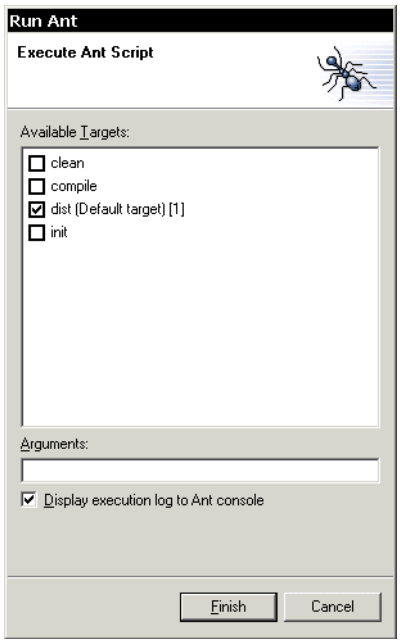


Figure 28-5 Running Ant

In the Ant Console the generated classpath has been used to compile the Java code. Notice that the path is including runtime-workspace which means it is using the runtime workspace, and not the host workspace, because we are in the test environment.

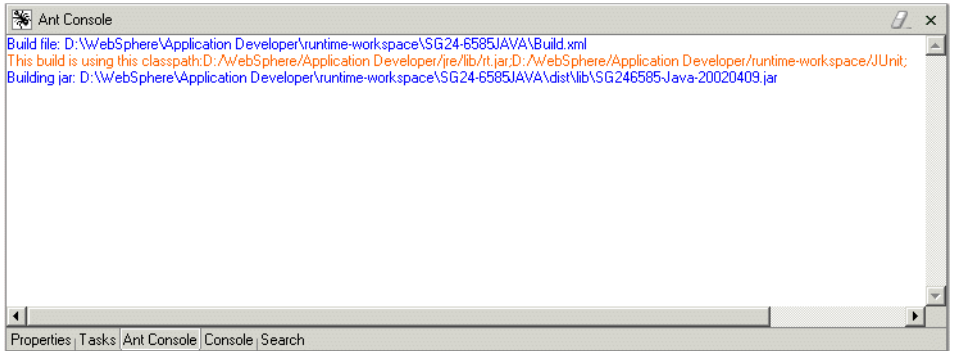


Figure 28-6 Ant console

29

Deploying a plug-in

During the design phase, plug-ins in your workspace are used as-is, so that you can quickly test and debug them. Once you reach the stage where you are satisfied with your code, you need to publish them in a form suited for delivery on the target platform.

This chapter describes following topics.

- Deploying a plug-in
- Installing a plug-in
- Creating a fragment for the existing plug-in
- Creating a component

© Copyright IBM Corp. 2002

29.1 Publishing a plug-in

There are two ways to publish your plug-in:

- **Packaging your plug-in(s) into a component**: This lets you use the platform Install and Update Manager to deliver your component. You can make it available on a server and publish the URL. The platform Update Manager can be used to download and install your component.
- **Build a JAR**: Your users will need to copy the JAR and `plugin.xml` to their platform installation directory.

29.1.1 Building a plug-in JAR

To build a plug-in JAR, select the plugin.xml file for your plug-in. Then select **Create plug-in JARs** from the context menu (Figure 29-1). A wizard will guide you through the creation of a build script for packaging your JARs.

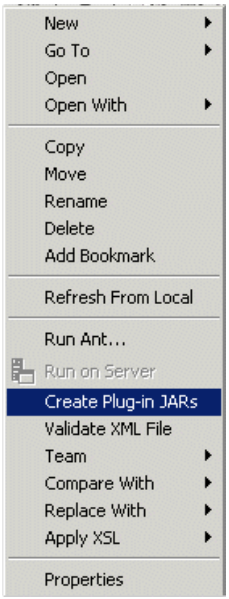


Figure 29-1 Create a plug-in JAR

In the plug-in build wizard, you can choose to build an ant script and build a plug-in jar. The plug-in jar is your actual publishing jar, and the ant script is for rebuild use. In this case, check both and click **Finish** (Figure 29-2).

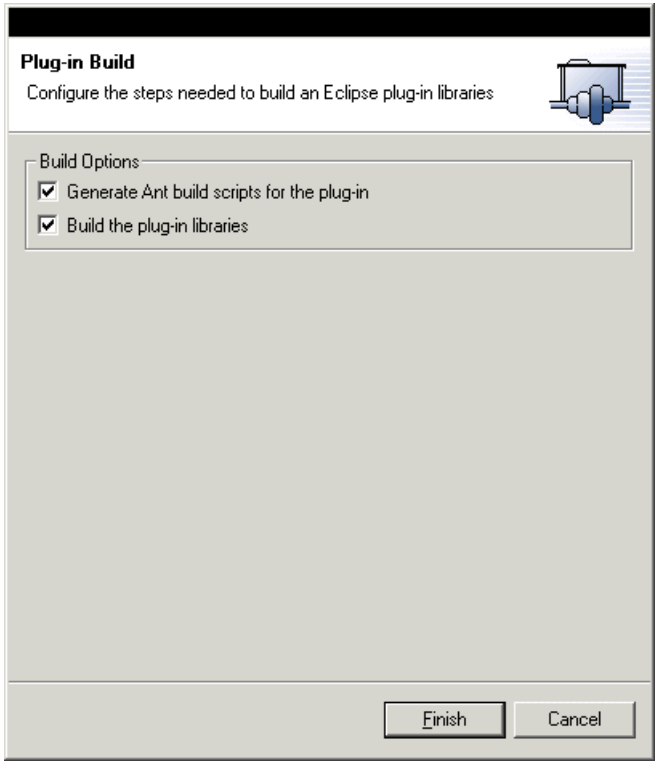


Figure 29-2 Plug-in build wizard

Once you created the jar files, the following files are added to your project.

- `build.xml`
- `ITSOJarEditor_1.0.0.zip`
- `ITSOJarEditor.jar`
- `ITSOJarEditor.jar.bin.log`

The `build.xml` is the build script to build this plug-in (Figure 29-3). You can rebuild it using this ant script.

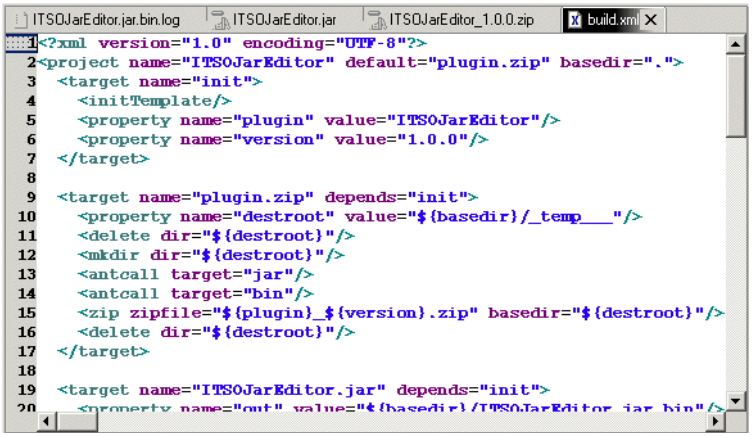


Figure 29-3 Build.xml for ITSOJarEditor project

The `ITSOJarEditor_1.0.0.zip` is a deliverable file. The `_1.0.0` part of its name is the version. It contains plugin.xml, `ITSOJarEditor.jar`. Those are required to be able to install this plug-in. Figure 29-4 shows the zip file contents using ITSOJarEditor.

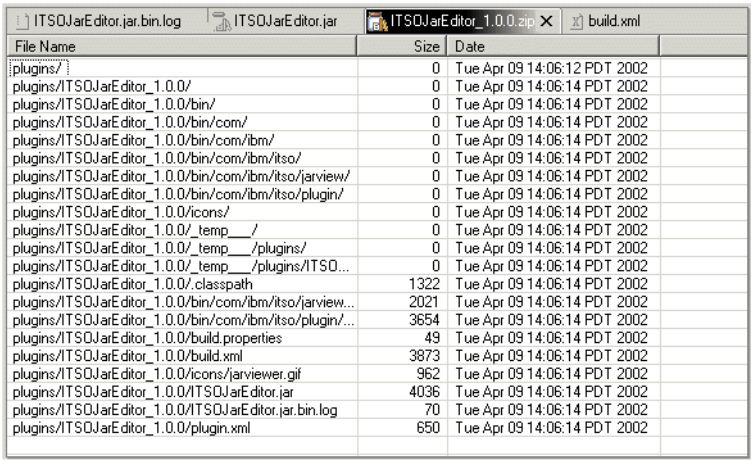

ITSOJarEditor.jar.bin.log
ITSOJarEditor.jar
ITSOJarEditor_1.0.0.zip
build.xml
File Name
Size
Date
plugins/ 0 Tue Apr 09 14:06:12 PDT 2002
plugins/ITSOJarEditor_1.0.0/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ibm/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ibm/itso/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ibm/itso/jarview/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ibm/itso/plugin/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/icons/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/_temp___/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/_temp___/plugins/ 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/_temp___/plugins/ITSO... 0 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/.classpath 1322 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ibm/itso/jarview... 2021 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/bin/com/ibm/itso/plugin/... 3654 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/build.properties 49 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/build.xml 3873 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/icons/jarviewer.gif 962 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/ITSOJarEditor.jar 4036 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/ITSOJarEditor.jar.bin.log 70 Tue Apr 09 14:06:14 PDT 2002
plugins/ITSOJarEditor_1.0.0/plugin.xml 650 Tue Apr 09 14:06:14 PDT 2002

Figure 29-4 ITSOJarEditor_1.0.0.zip contents

Your users will need to manually place the JARs and the `plugin.xml` into the plug-in's directory beneath the platform plug-ins directory.

The `ITSOJarEditor.jar` is the actual plug-in jar; Figure 29-5 and `ITSOJarEditor.jar.bin.log` is the build log file.

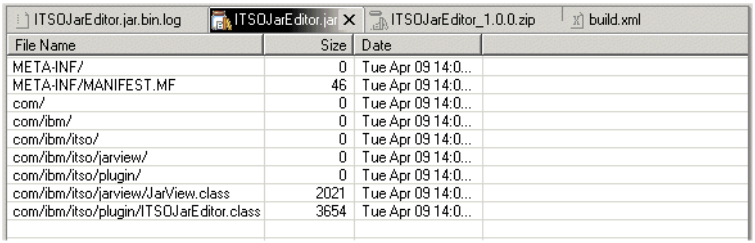


Figure 29-5 ITSOJarEditor.jar contents

Export your deliverable using export tool and export as a file system.

29.2 Installing the plug-in

Once you created a deliverable you can test its installation.

Unzip `ITSOJarEditor_1.0.0.zip` file into the Application Developer directory. Notice that the jar file's icon has been changed to jar icon, and it has **Open with Jar Editor menu**. Now you can use the ITSOJarEditor (Figure 29-6).

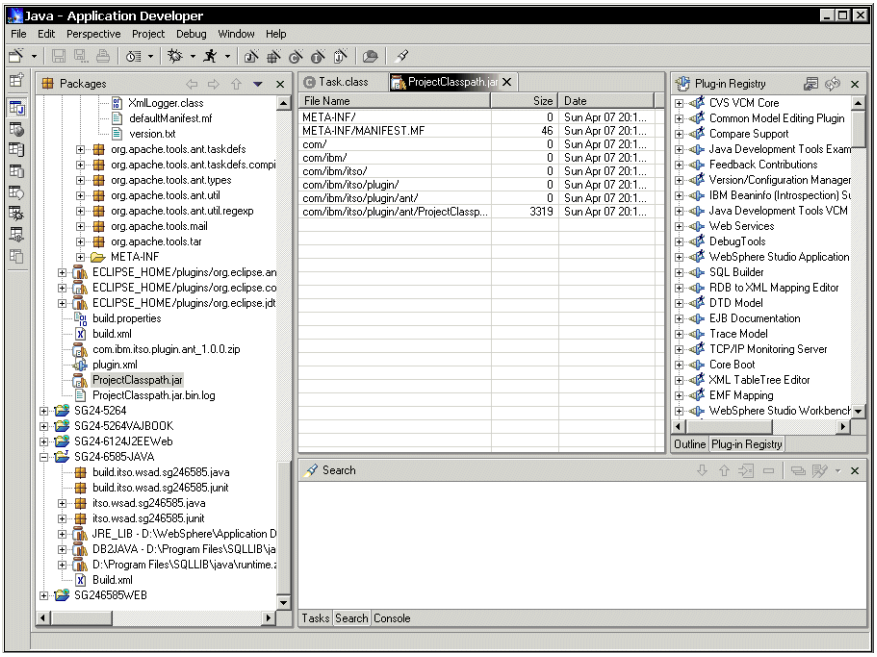


Figure 29-6 Executing with the plug-in

If the plug-in is not working check the **Plug-in registry** from **Perspective** menu. If it installed correctly, the ITSO Jar Editor plug-in should be there with running icon (Figure 29-7).

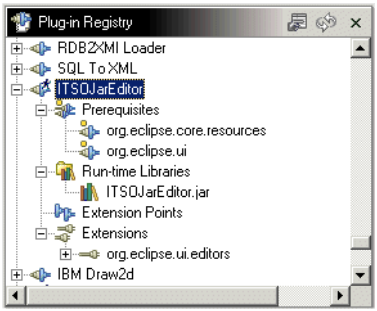


Figure 29-7 Plug-in registry

29.3 Fragment development

A **plug-in fragment** is used to provide additional plug-in functionality to an existing plug-in after it has been installed. Fragments are ideal for shipping features like language or maintenance packs that typically trail the initial products by a few months. When a fragment is detected by the platform and its "target plug-in" is found, the function in the fragment is merged with the original function in the target plug-in. If you query the plug-in registry, you will see the features defined in a fragment as if they were contributed by the original plug-in.

While this merging mechanism is good from a runtime point of view, developers need to view fragments as separate entities while working on them. Fragment development is often done by different teams, on a different schedule, sometimes even on different operating systems than the original plug-in.

PDE provides full support for fragment development. Fragments can be viewed as "limited plug-ins." They have all of the capability of regular plug-ins except the ability to establish dependencies on other plug-ins. The only dependency they can have is on the target plug-in itself. This dictates the classes that fragments can see and extension points they can extend.

The PDE concept of workspace and external plug-ins turns out to work quite nicely when developing a fragment. You can work on a fragment whose target is an external plug-in. Since external plug-ins cannot be changed inside the workbench, the environment inherently supports the fact that the fragment should be developed without modifying its target plug-in.

29.3.1 Writing a fragment for ITSOJarEditor

The PDE wizards and editors that manipulate plug-ins and fragments are very similar. You must, however, be aware of some important differences.

You start by creating a new **Fragment project** Figure 29-8.

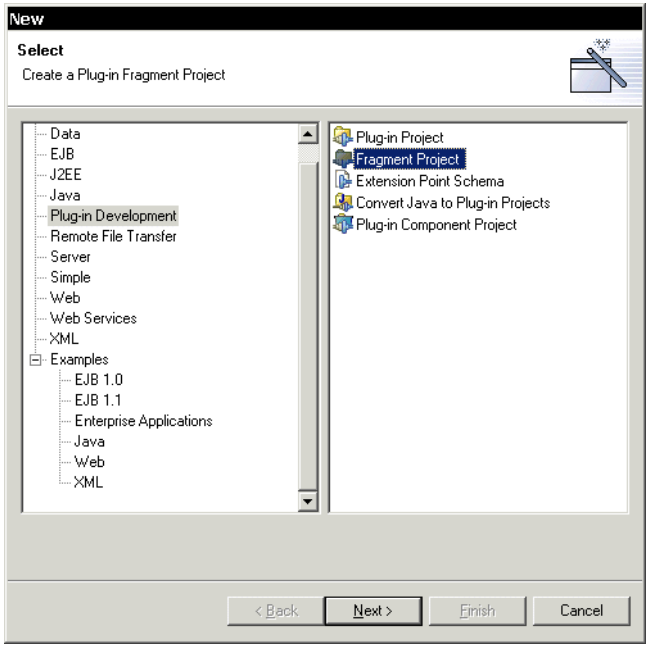


Figure 29-8 Create new Fragment project

On the first page of the New Fragment Wizard, type the project name `ITSOJarEditorJ`. Click **Next** and accept the default values on the second page (Figure 29-9).

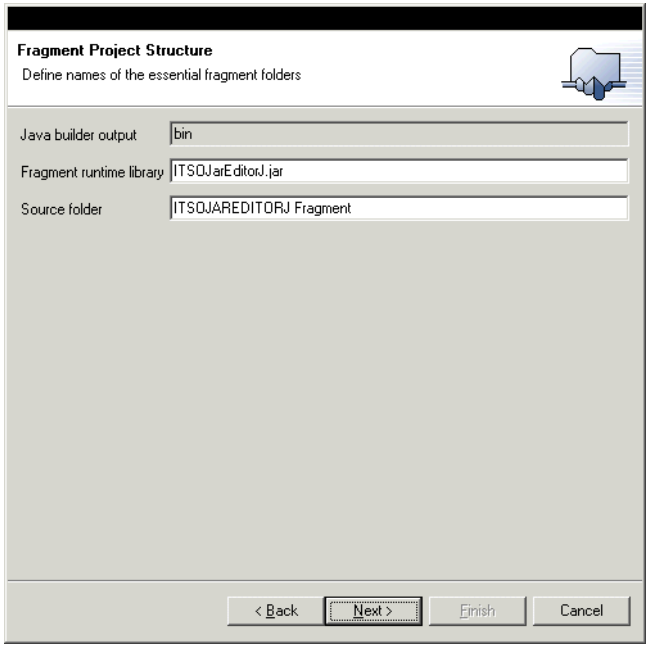


Figure 29-9 Fragment project structure

On the Fragment Code Generators page, select the radio button for creating a fragment from a template and select the Default Fragment Generator wizard. After clicking **Next**, you should see the Simple Fragment Content page (Figure 29-10).

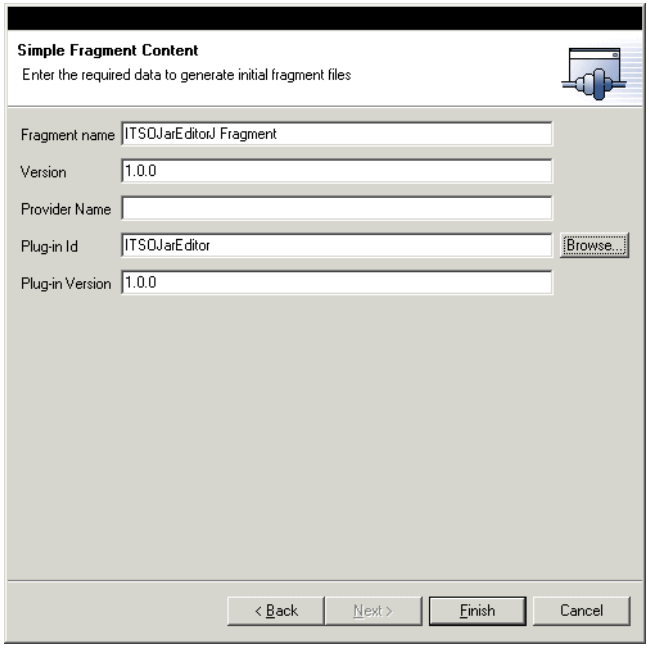


Figure 29-10 Simple Fragment content

This page looks a little different from the one in plug-in wizard. It has two additional fields: (target) plug-in ID and (target) plug-in version. Since we are writing a fragment for a specific plug-in, we can use the **Browse** button to select ITSOJarEditor from the Workspace Plug-ins group (we could also pick any of the external plug-ins) (Figure 29-11).

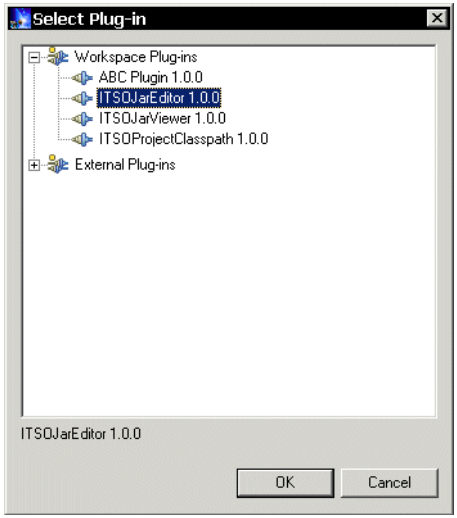


Figure 29-11 Select the target plug-in

It is almost identical to the plug-in manifest editor with the following exceptions:

- In the Overview page, the **class** attribute is gone. Fragments do not have a plug-in class since they follow the life cycle of their target plug-in. Instead, the target plug-in id and version fields are shown.
- There is no Dependencies page. If you select the fragment project and select **Properties—>Build Path** you will see that the fragment has the identical build path as the target plug-in.

Changing the menu language

We will add a similar editor extension as the ITSOJarEditor, but this time in japanese (use the language of your choice). Go to the Extensions page in the Fragment Manifest editor. Click **New** to launch the Extension wizard. Select **Generic Wizards** and **Schema-Based Extensions**. Click **Next**. Select **Editors** from the list of extension points. Click **Finish**.

Select the Internal and External Editors and then select **New—>Editor** from the context menu. Move to the property sheet and change the name property to `Jar Editor` into the one in your chosen language (Figure 29-12).

Select the class property for editing and bring up the Cell Editor dialog. Enter a name of an existing Java class radio button and click **Browse.** Then select the ITSOJarEditor class (at this time, we will reuse this class).

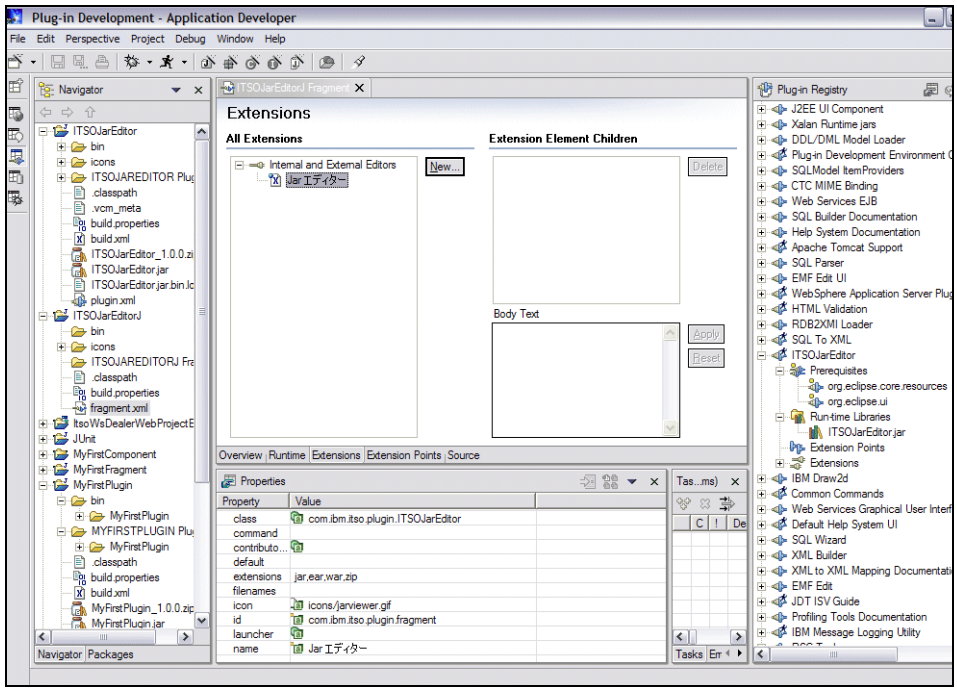


Figure 29-12 Editing the manifest

Setup the **id**, **icons**, and **extensions** properties as `com.ibm.itso.plugin.fragment`, `icons/jarviewer.gif, and jar,ear,war,zip` respectively.

Save and close the Java editor and Fragment Manifest editor.

When you run the fragment using the **Run** icon, the run-time platform instance should now have the new Editor available (Figure 29-13).

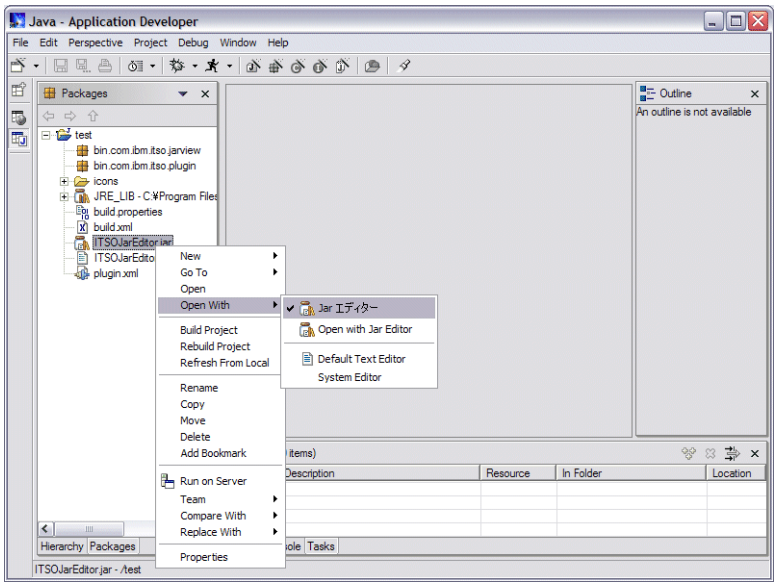


Figure 29-13 Running the fragment

29.3.2 Building a fragment

Fragments can be deployed in a similar way to plug-ins.They can be delivered as part of a platform component or in a fragment JAR. To build the fragment JAR, select the `fragment.xml` file and select **Create fragment JARs** from its context menu (Figure 29-14). A wizard will guide you in creating a build script to produce all the required JARs. The JARs and the `fragment.xml` should be manually copied to the target platform plug-in directory.

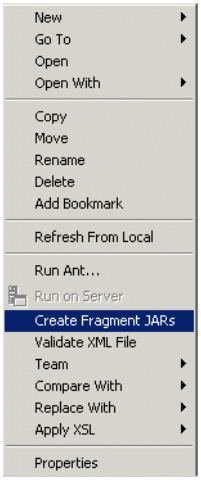


Figure 29-14 Create fragment JARs

29.4 Build configuration

The component build mechanism is driven by a build configuration. The build configuration for an individual plug-in, fragment, or component is found in a build.properties file for the corresponding element.

PDE supplies wizards that create the `build.properties` file for each plug-in and fragment that is to be packaged into a component (Figure 29-15). The `build.properties` file contains information on how to compile source folders into JARs. Typically, you modify this file inside the component manifest editor. You can also add entries directly to the file using another editor.

PDE provides a simple editor for `build.properties` that has form and source views. The file itself follows the Java properties format. You need to provide a number of keys and their corresponding values. Multiple values are separated using a comma as the delimiter.

There are six points where text can be specified by a plug-in or fragment for inclusion in the build. These points are:

- `bin.includes`
- `bin.excludes`
- `javadoc.packages`
- `javadoc.excludedpackages`
- `src.includes`
- `src.excludes`

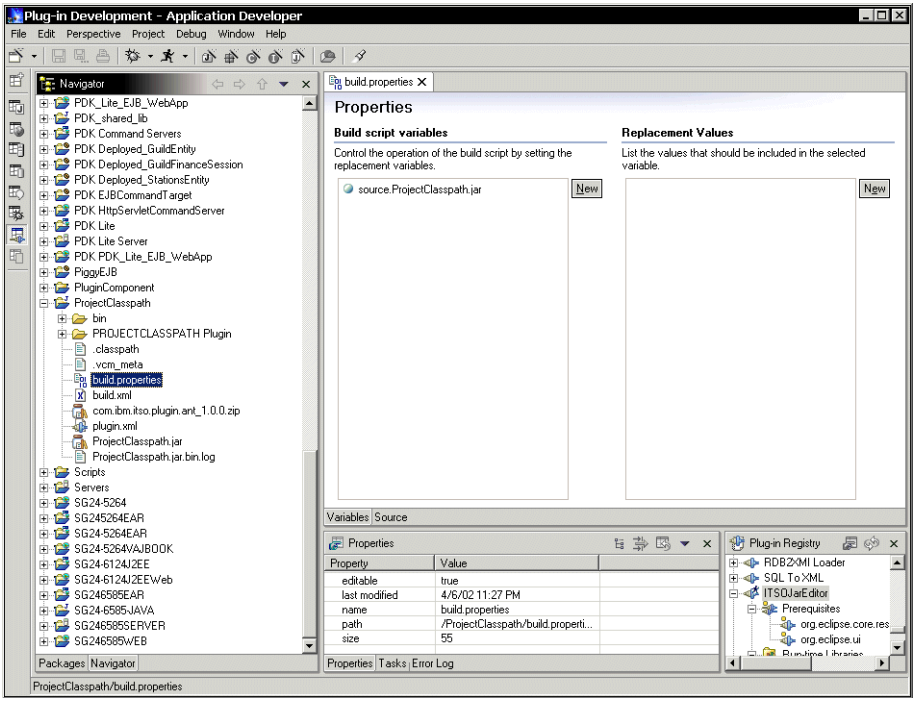


Figure 29-15 Build.properties

The values defined in these points are expressed as Ant "patterns". Typically this is to a comma-separated list of strings to match. Standard formats give the expected results. For example, "`.jar`" indicates all jar files in the top level directory. The patterns are not deep by default. If you want to describe all Java files for example, you should use the pattern "`**/*.java`". The pattern "`**`" matches any number of directory levels. Similarly, to describe whole sub trees, use "`foo/`". Default values for these points are always automatically generated with identifiers of form `auto.<lastPortionOfPointId>`. This is done to allow your override to extend these default values if desired. To set override values, a plug-in defines additional name/value properties in the previously-mentioned build.properties file.

29.5 Creating a component

The platform is designed to accept updates and additions to the initial installation. The platform Update Manager handles this task by connecting to sites where updates are posted. You need to package your work in a form that will be accepted by the Update Manager. When you deliver an update to the platform, you are contributing a component.

Components consist of a manifest that provides basic information about the component and its content.

Note: Components may include plug-ins, fragments and any other files that are important for the component. The delivery format for a component is a JAR.

In PDE, the typical development process looks like this:

1. Projects for plug-ins and fragments are created.
2. Code for plug-ins and fragments is created, tested and debugged.
3. When you want to make your code available to others, you create a new component project.
4. Individual build properties for each plug-in and fragment are tailored to control what files are included and excluded from the packaging.
5. Versions are synchronized with previous versions of the component, so that the Update Manager will know that a component is a newer version of an already installed component.
6. The component JAR is built.
7. The component manifest and component JAR are published on the update server and made available for download.

29.5.1 Setting up a component project

Note: Similar to plug-ins and fragments, PDE treats platform components as projects.

PDE attaches a special **component** nature to these projects to differentiate them from other project types. The project must have a specific folder structure and a component manifest. The project must be set up with references to all of the plug-in and fragment projects that will be packaged into the component.

PDE provides a wizard for setting up a component project. Typically you use this wizard to set up a component once you are done developing plug-ins and fragments. However, you can create the component at any stage of development and add new plug-ins later.

Assuming that you have followed the previous examples, you should have `ITSOJarEditor` plug-in and `ITSOJarEditorJ` fragment in your workspace already. We will create a sample component and package these artifacts to be ready for delivery.

29.5.2 Creating a component project

We start by creating a new component project (Figure 29-16).

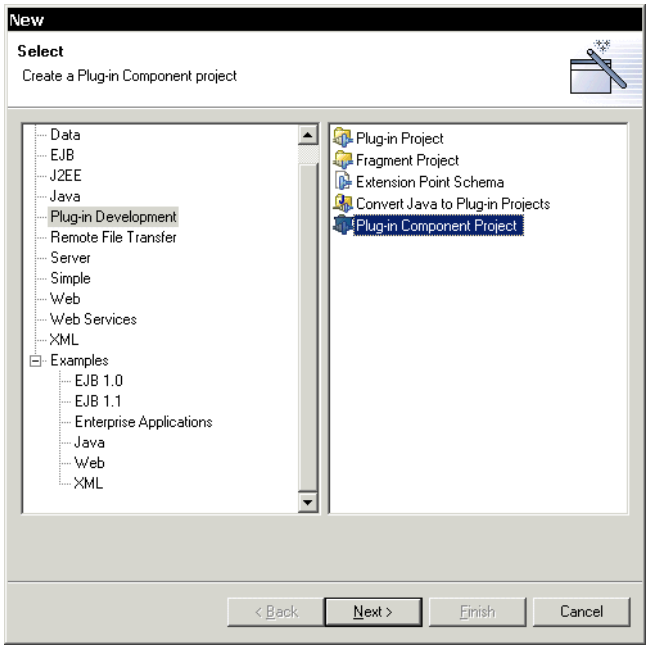


Figure 29-16 Start new Component project

Set the name of the project to `ITSOJarEditor Component` and click **Next** (Figure 29-17).

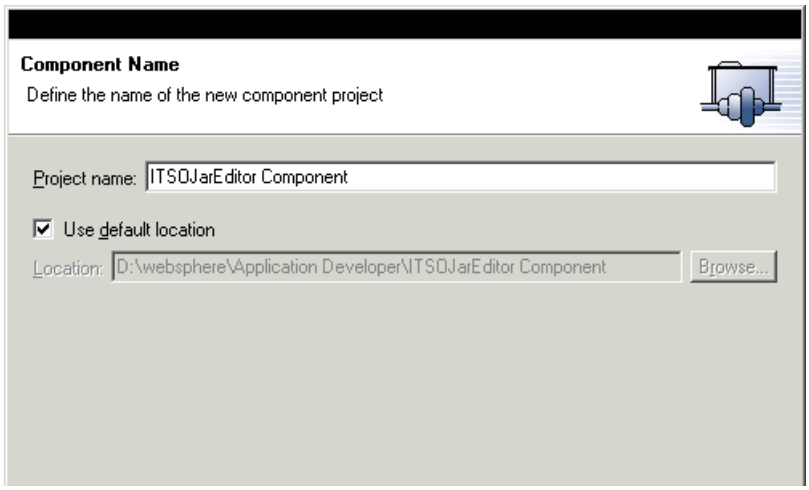


Figure 29-17 Component name

Set the **Component ID** to `com.ibm.itso.plugin` and the **Component Version** to `1.0.0`. Set the **Component Provider** to `IBM ITSO`, and enter a description for the component (Figure 29-18).

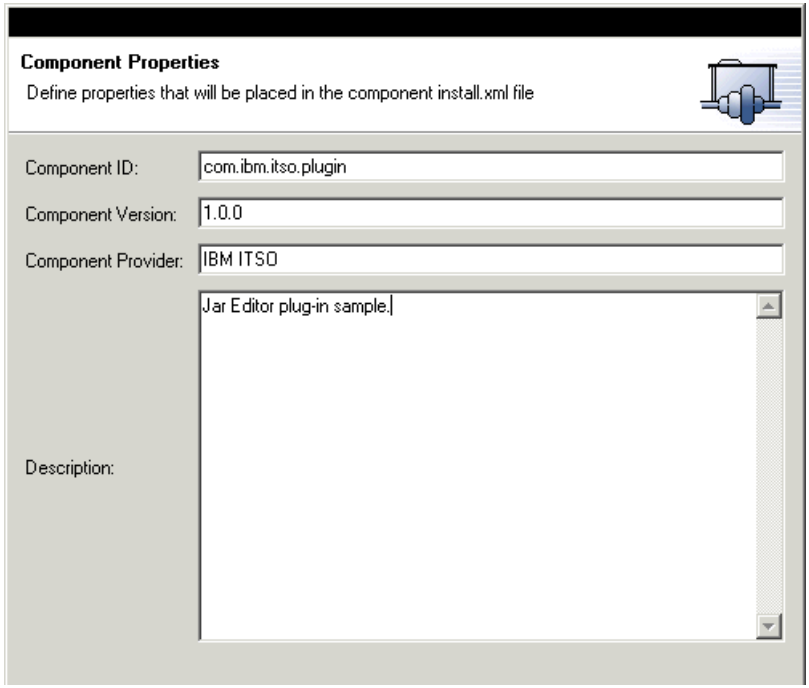

Component Properties
Define properties that will be placed in the component install.xml file
Component ID: com.ibm.itso.plugin
Component Version: 1.0.0
Component Provider: IBM ITSO
Description: Jar Editor plug-in sample.

Figure 29-18 Component properties

In the following page, check the `ITSOJarEditor` and `ITSOJarViewer`, (we are going to deliver these two plug-ins) and click **Next** (Figure 29-19).

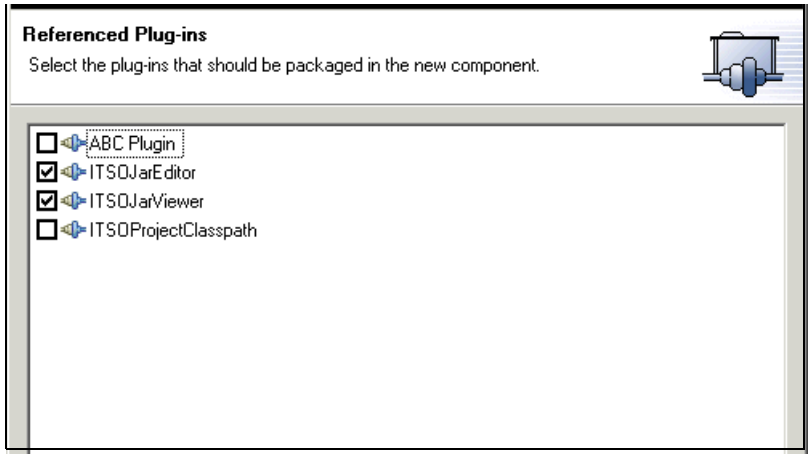


Figure 29-19 Referenced plug-ins

On the following page, Figure 29-20, we only have one fragment and we want to deliver it as well. So check it and click **Finish**.

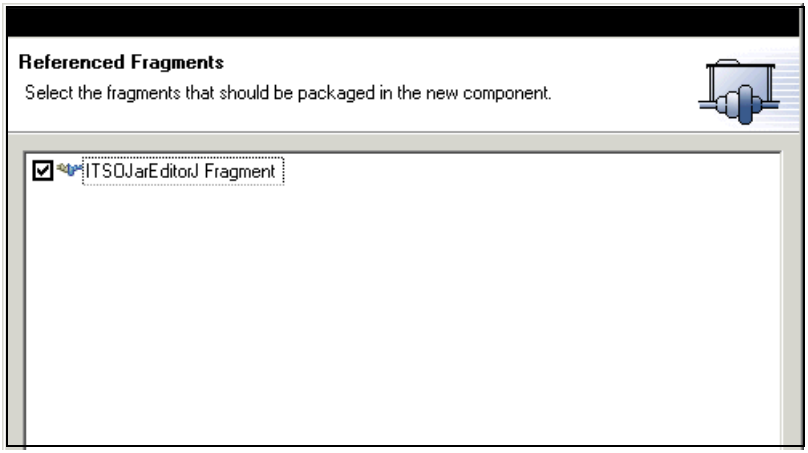


Figure 29-20 Referenced fragments

You should now have the ITSOJarEditor Component project in your workspace. The project should have a folder structure with the path `install/components/com.ibm.itso.plugin_1.0.0`. This structure should contain the component manifest file and `install.xml`. The component manifest editor will open for you (Figure 29-21).

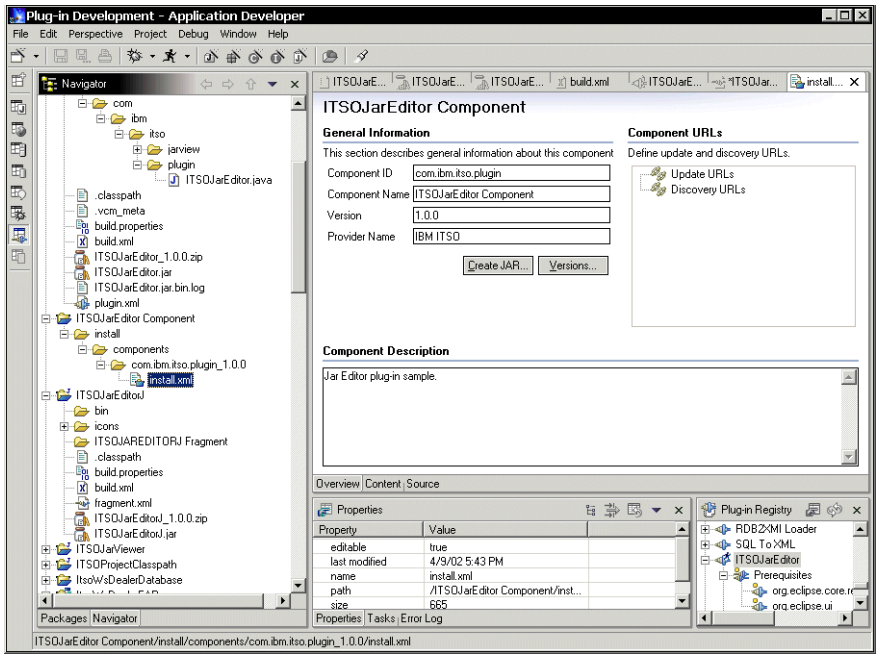


Figure 29-21 ITSOJarEditor component

The component manifest editor uses the same concepts you have seen in the other PDE editors.

It has two form pages, (Overview and Content), and a Source page that shows the raw XML code of the manifest file. Information that is entered during the component project setup can be changed anytime later on the Overview page.

In addition, you can provide URLs for update sites and discovery sites in the property sheet. The update site URLs are shown to users by the Update Manager when they update the platform.

The discovery site URLs are used to point users to other interesting components and/or sites.

Plug-ins and fragments that are part of your component are listed in the Content page. The check boxes in the lists represent all of the valid plug-ins and fragments in the workspace. By selecting the check boxes, you are instructing the editor to include them in the component. If you double-click on a plug-in or fragment entry, the plug-in or fragment manifest editor will be opened for you on the selected item (Figure 29-22).

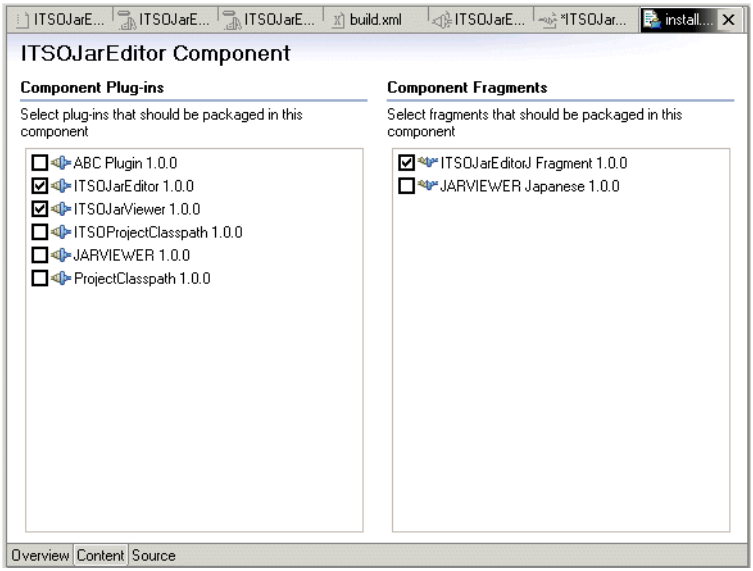


Figure 29-22 Plug-ins and fragments

29.5.3 Synchronizing versions

The versions of plug-ins and fragments should be synchronized with the version of the packaged component so that you can manage plug-in, fragment, and component versions consistently.

Developers typically ignore individual manifest versions until it is time to deploy their components.

The Update Manager uses component versions to determine whether a plug-in is older or newer than one already installed. Plug-ins and fragments need to follow the same version number conventions so there is no confusion about which plug-in version belongs to which component version.

The most convenient way to synchronize versions is to pick the version of the component and force it into all the plug-ins and fragments that the component references. This operation updates manifest files, so you are required to close all the manifest editors before proceeding (Figure 29-23).

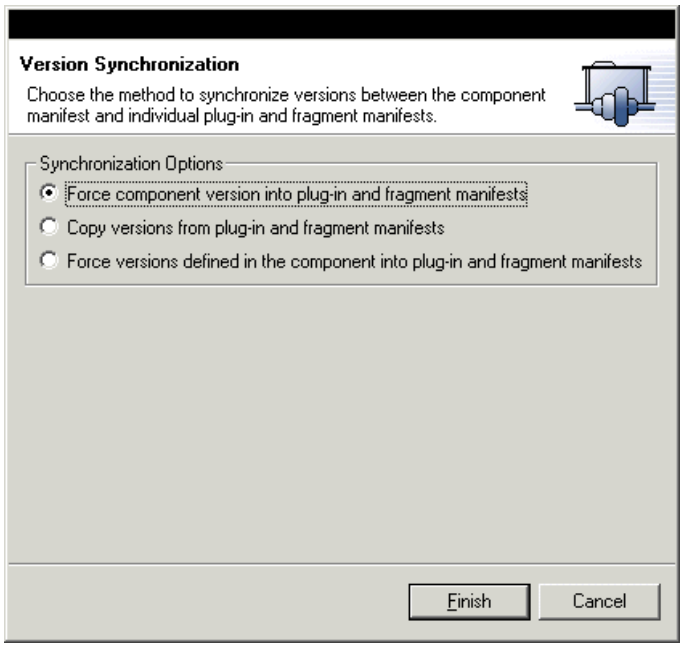

Version Synchronization
Choose the method to synchronize versions between the component manifest and individual plug-in and fragment manifests.
Synchronization Options
Force component version into plug-in and fragment manifests
Copy versions from plug-in and fragment manifests
Force versions defined in the component into plug-in and fragment manifests
Finish
Cancel

Figure 29-23 Synchronizing versions

29.5.4 Generating a component JAR

Once versions are synchronized, you can package your component in the format fit for publishing. To build the Component JAR, select the `install.xml` file and then select **Create Component JAR...** from the context menu. A wizard will guide you through the creation of a build script to produce all the required JAR files (Figure 29-24).

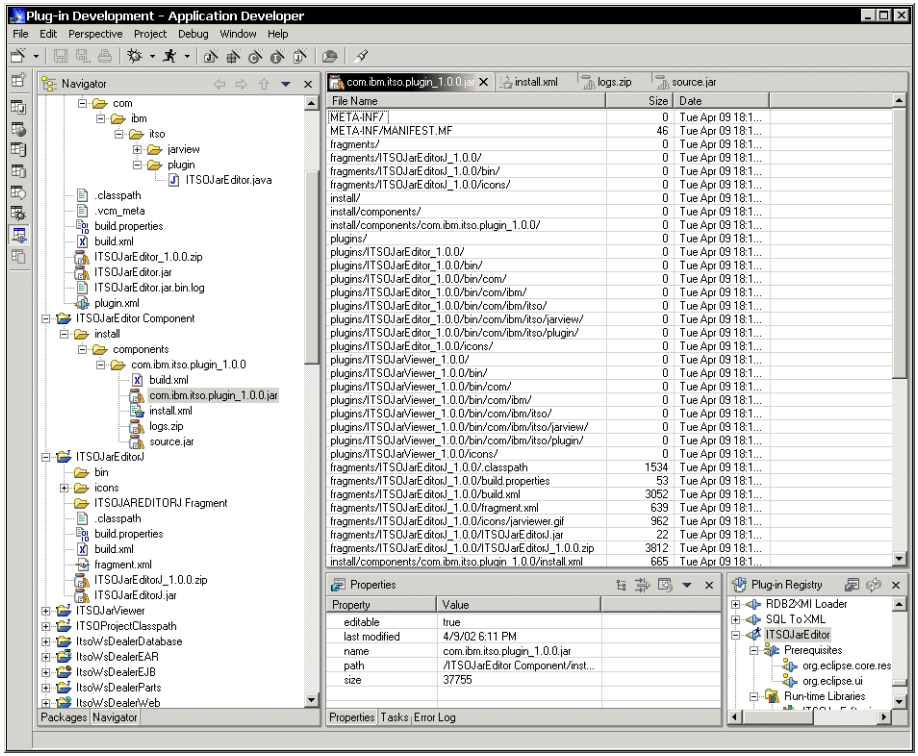


Figure 29-24 Created component

29.6 Publishing a JAR

PDE does not provide automated publishing support for components.

To publish a component you need to place the `install.xml` and the component JAR on a platform update server. You simply take your component manifest and the JAR and place them in a directory that has the same name format as the JAR itself minus the `.jar` extension. You need to add an entry in the `install.index` file because Update Manager cannot simply iterate through the directories when accessing the server using the HTTP protocol.

If everything is done correctly, products built on the platform can use Update Manager to point to your site, install your component, and restart. When they restart the platform, your plug-in written in PDE should be available.

As you continue work on your component, you can create newer versions and periodically publish them on the update server. Just don't forget to increment version numbers.

Part 9

Appendixes

© Copyright IBM Corp. 2002

A

Installing WebSphere Studio Application Developer

In this appendix we describe how to install WebSphere Studio Application Developer. The installation instructions specifically apply to Version 4.0.2., but also should be very similar for more recent versions of the product.

© Copyright IBM Corp. 2002

Things to do before installation

Before you proceed with installation, please verify that your hardware and software configurations meet the following prerequisites:

- Windows 2000, Windows ME, Windows 98, or Windows NT 4.0 with Service Pack 6a or higher
- Microsoft Internet Explorer 5.5 or higher
- TCP/IP installed and configured
- A mouse or alternative pointing device
- Pentium II processor or higher recommended
- SVGA (800x600) display or higher (1024x768 recommended)
- 256 MB RAM minimum (512 MB recommended)
- A minimum of 400 MB free disk space based on NTFS. Actual disk space on FAT depends on hard disk size and partitioning

Also please check the following before beginning the install:

- In addition to the disk space requirements for the product, you need to have at least 50 MB of space available on your Windows system drive and the TMP or TEMP environment variable must be pointing to a directory with at least 10 MB free.
- If you have the IBM HTTP server or WebSphere Application Server running, they need to be shut down.
- The Services window should not be open. If it is, the Remote Agent Controller cannot be installed.

Note: If you have VisualAge for Java or any version of WebSphere Studio already installed, there is no need to un-install before installing Application Developer.

Installing WebSphere Studio

Start the installation by running SETUP.EXE from the directory where the WebSphere Studio code is.

After the first couple of panels where you have to accept license information and choose a directory where to install WebSphere Studio Application Developer, you will be prompted to select your primary user role (Figure A-1).

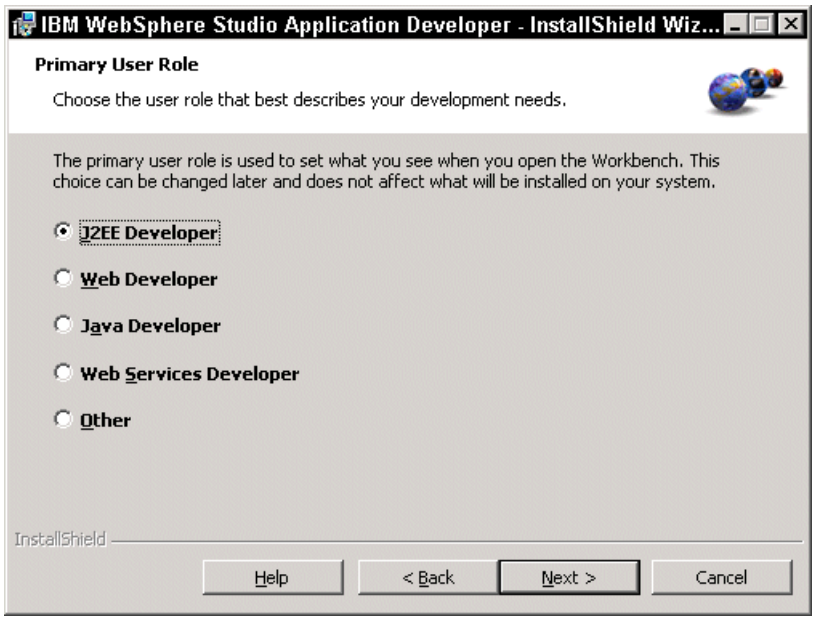

IBM WebSphere Studio Application Developer - InstallShield Wiz...
Primary User Role
Choose the user role that best describes your development needs.
The primary user role is used to set what you see when you open the Workbench. This choice can be changed later and does not affect what will be installed on your system.
J2EE Developer
Web Developer
Java Developer
Web Services Developer
Other
InstallShield
Help
< Back
Next >
Cancel

Figure A-1 Select primary user role

On this page you select what your primary role will be. This choice will determine what default perspective you will first see when you open the Application Developer Workbench. Once you have installed the product you can change this preference.

On the next page you select which SCM tool you want to work with (Figure A-2).

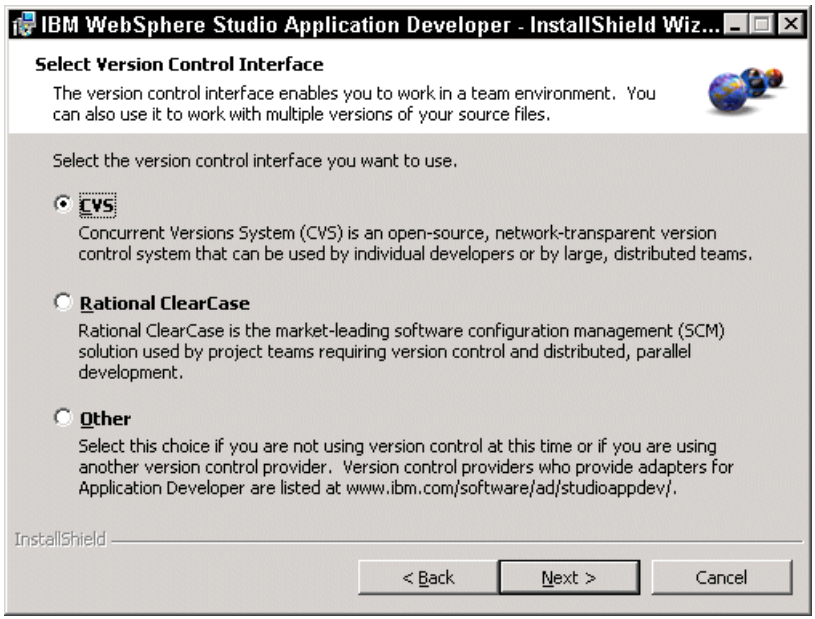

IBM WebSphere Studio Application Developer - InstallShield Wiz...
Select Version Control Interface
The version control interface enables you to work in a team environment. You can also use it to work with multiple versions of your source files.
Select the version control interface you want to use.
CVS
Concurrent Versions System (CVS) is an open-source, network-transparent version control system that can be used by individual developers or by large, distributed teams.
Rational ClearCase
Rational ClearCase is the market-leading software configuration management (SCM) solution used by project teams requiring version control and distributed, parallel development.
Other
Select this choice if you are not using version control at this time or if you are using another version control provider. Version control providers who provide adapters for Application Developer are listed at www.ibm.com/software/ad/studioappdev/.
InstallShield
< Back
Next >
Cancel

Figure A-2 Select version control interface

The predefined choices are currently CVS and Rational ClearCase. If you are not planning to use any versioning system at this time, or if you will be using another third-party one, select the **Other** radio button.

Important: The CVS and ClearCase *server code* is not installed as part of the WebSphere Studio installation. If you select either of these two, the code will have to be installed separately once the Application Developer installation is complete.

For more information about team development and Version Control Systems, see Chapter 23, “Version control” on page 483.

Click **Install** on the final page to start the installation.

Verifying the installation

Once the installation program has run, click the Windows **Start** button and select **Programs—>IBM WebSphere Studio Application Developer—>IBM WebSphere Studio Application Developer**.

If the installation has worked correctly you should see your default perspective with the Application Developer welcome page (Figure A-3).

Note: The figure shows that the Java perspective has been opened. This is because Java developer was selected as the user role when Application Developer was installed.

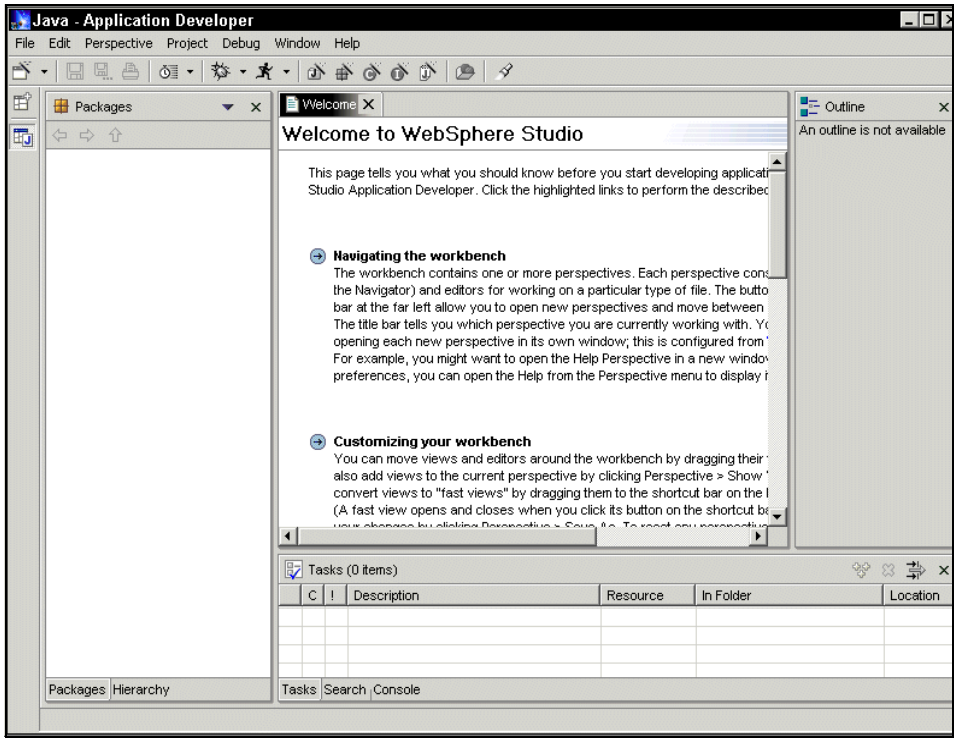

Java - Application Developer
File Edit Perspective Project Debug Window Help
Packages
Welcome
Welcome to WebSphere Studio
This page tells you what you should know before you start developing applicati
Studio Application Developer. Click the highlighted links to perform the describec
Navigating the workbench
The workbench contains one or more perspectives. Each perspective cons
the Navigator) and editors for working on a particular type of file. The butto
bar at the far left allow you to open new perspectives and move between
The title bar tells you which perspective you are currently working with. Y
opening each new perspective in its own window; this is configured from
For example, you might want to open the Help Perspective in a new windo
preferences, you can open the Help from the Perspective menu to display
Customizing your workbench
You can move views and editors around the workbench by dragging their
also add views to the current perspective by clicking Perspective > Show
convert views to "fast views" by dragging them to the shortcut bar on the
(A fast view opens and closes when you click its button on the shortcut ba
Outline
An outline is not available
Tasks (0 items)
C ! Description Resource In Folder Location
Packages Hierarchy
Tasks Search Console

Figure A-3 Application Developer default perspective

B

Installing IBM WebSphere Application Server 4.0 AEs

In this appendix we describe how to install IBM WebSphere Application Server 4.0 Advanced Edition Single Server (AEs).

© Copyright IBM Corp. 2002

Things to do before installation

Prior to installing AEs, the following checks and tasks need to be completed on the WebSphere server machine:

1. Check hardware and software prerequisites.
2. Create groups and users.
3. Check that IP ports are unused.
4. Stop the Web server processes.

Hardware and software prerequisites

AEs has the following hardware and software requirements.

- Hardware
  - 180 MB diskspace (minimum) for AEs
  - 50 MB diskspace (minimum) for IBM HTTP Server
  - 135 MB diskspace (minimum) for TEMP directory
  - 500Mhz Pentium
  - 384 MB RAM minimum, 512 MB recommended
  - Ethernet or Token Ring card
  - CD-ROM drive
  - Network connectivity to the Internet
- Software
  - Microsoft Windows 2000 Server, SP 1 or 2 *or* Microsoft Windows NT Server 4.0 SP 6a
  - IBM HTTP Server 1.3.19

(IBM HTTP Server is included with AEs and will be installed if not already present.)

Create groups and users

To create the required groups and users, perform the following steps:

1. If you have not already done so, create a Windows 2000 user under which the WebSphere service will be run, as follows
   - Locally defined (not a member of a Windows domain)
   - Member of administrator's group

You can create local users and assign group memberships by clicking **Control Panel—>Administrative Tools—>Computer Management —>System Tools —>Local Users and Groups**.

2. Assign the following rights to this user:
   - Act as part of the operating system
   - Log on as a service

You can assign user rights by clicking **Control Panel—> Administrative Tools —>Local Security Policy—>Local Policies—>User Rights Assignment**.

Tip: We suggest creating the user “wsadmin”.

Check that IP ports are unused

To check that the required ports are not in use, perform the following steps:

1. Check that there are no existing active services that use the following IP ports on the server:
   - 900 (bootstrap port)
   - 9000 (Location Service Daemon)
   - 9080 (default application server)

Use the following command for this task: `C:\> netstat -an`

Stop the Web server processes

The IBM HTTP Server process must be stopped while AEs is being installed. The installation changes the httpd.conf configuration file as part of the Web server plug-in component installation.

- Issue the command: `C:\> net stop "IBM HTTP Server"` or stop the service under **Control Panel—>Administrative Tools—>Services**.

Install WebSphere

To install AEs using the GUI installer interface, complete the following steps on the WebSphere server machine:

Tip: The WebSphere installer (setup.exe) also provides a non-GUI, scripted or “silent” mode of operation. See product documentation for details.

1. Log on with a user ID that has administrator rights to the local server domain.
2. Insert the AEs CD.
3. Start the AEs installation by double-clicking Setup from the root of the CD.
4. In the **Choose Setup Language** window, select your national language from the drop-down menu (English is selected by default) and click **OK**.
5. In the **WebSphere Application Server Attention** window, read the warnings and then click **Next** to continue.
6. In the **Installation Options** window shown in Figure B-1, select **Typical Installation**, and then click **Next**. If you have some special requirements, you may want to select **Custom Installation** instead. This will give you the option to deselect certain options, for example the IBM HTTP Server and the JDK.

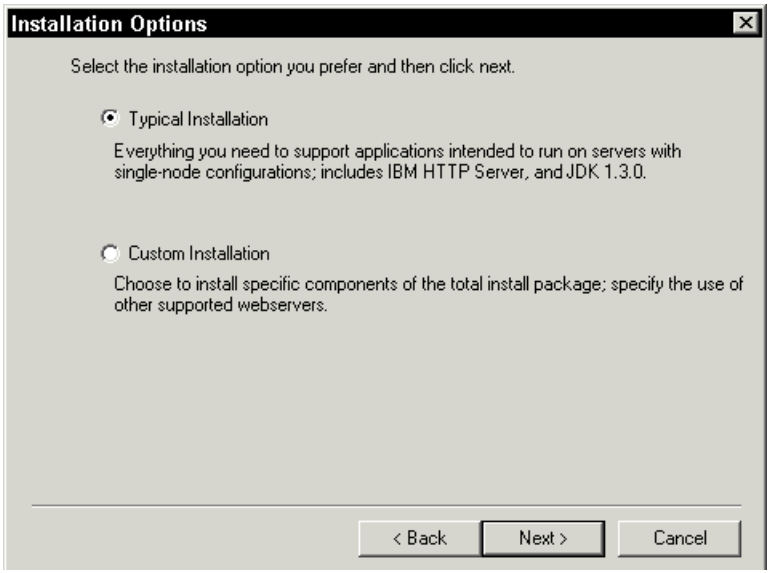

Installation Options
Select the installation option you prefer and then click next.
Typical Installation
Everything you need to support applications intended to run on servers with single-node configurations; includes IBM HTTP Server, and JDK 1.3.0.
Custom Installation
Choose to install specific components of the total install package; specify the use of other supported webservers.
< Back
Next >
Cancel

Figure B-1 Installation options

7. In the **Security Options** window, shown in Figure B-2, enter the user name and password of the Windows account under which AEs is to run as a service. This is the account created during the WebSphere pre-installation tasks. Click **Next** to continue.

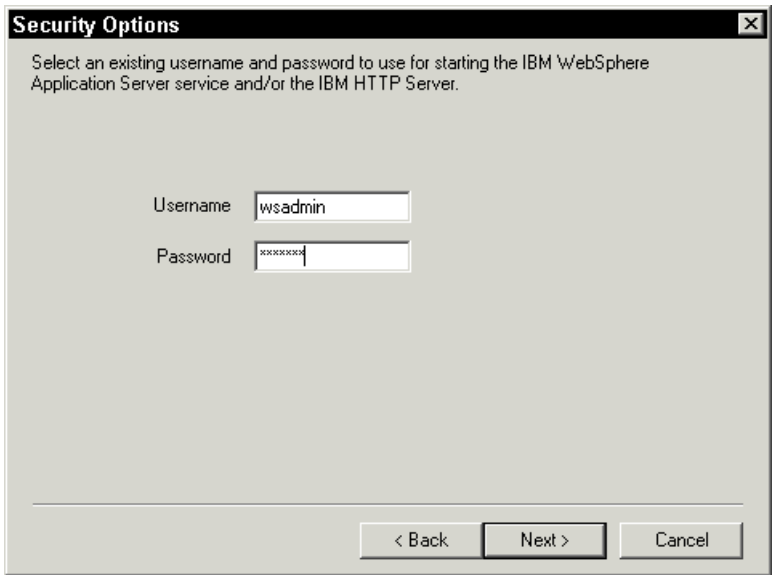

Security Options
Select an existing username and password to use for starting the IBM WebSphere Application Server service and/or the IBM HTTP Server.
Username
wsadmin
Password
< Back
Next >
Cancel

Figure B-2 Security options

8. In the **Product Directory** window, select the destination directory, and then click **Next**.
9. In the **Select Program Folder** window, accept the default and click **Next**.
10. In the **Install Options Selected** window, note the selected options and click **Next** to start the installation.
11. When the **Setup Complete** window is shown, click **Finish**.
12. In the **Restarting Windows** window, select to restart the computer to complete the installation.

Verifying the installation

Perform the following tasks to verify that the installation was successful:

1. Start the WebSphere administrative server processes.
2. Start WebSphere Default Server.

Start the WebSphere administrative server processes: The WebSphere administrative server needs to be started in order to test the installation.

1. Start the server by selecting **Start application server** from the WebSphere menu or by entering C:\WebSphere\AppServer\bin\startServer.bat on a command line.

2. The startup of WebSphere administrative server was successful if the last line of the <WAS_HOME>\logs\default_server_stdout.log file is similar to the following:

   ```
   [02.04.05 15:48:56:309 PST] 5dc79b14 Server A WSVR0023I: Server Default
   Server open for e-business
   ```

3. Verify that the Default Server Web container has been properly installed and configured by accessing its servlets through the Web server “embedded” within the WebSphere V4.0 Web container:

   a. Using a Web browser, request the following URL:

      ```
      http://localhost:9080/servlet/snoop
      ```

      A window similar to the one shown in Figure B-3 should be displayed in your browser.

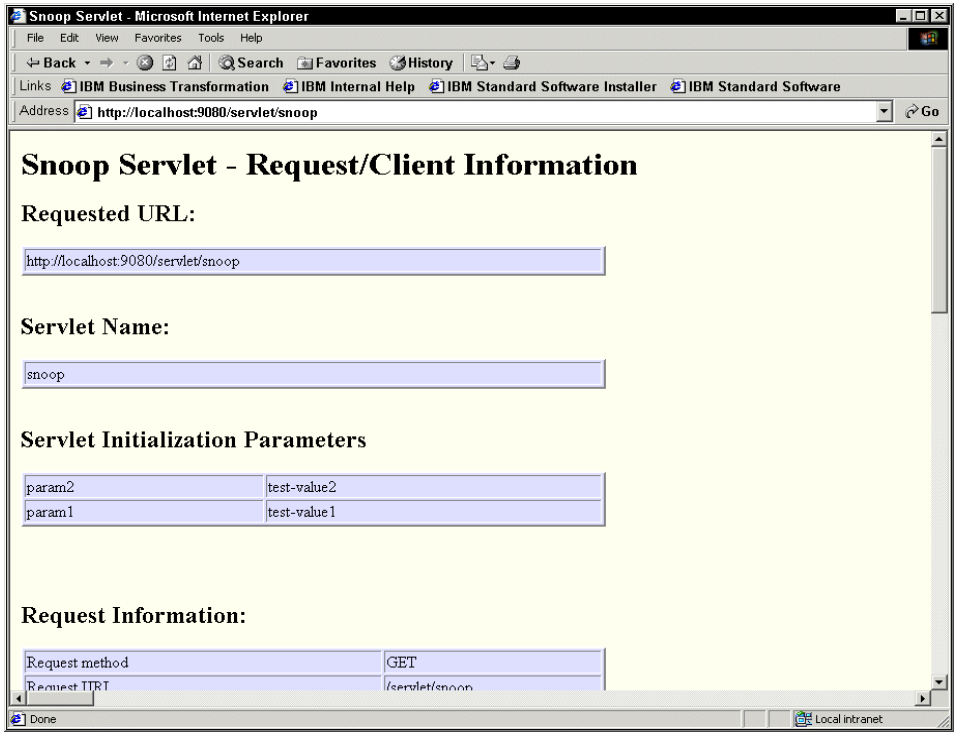


Figure B-3 Snoop servlet accessed through embedded Web server

WebSphere Application Server AEs has now been successfully installed on your workstation.

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described in the following sections.

© Copyright IBM Corp. 2002

Locating the Web material

The sample code associated with this redbook is available in soft copy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG246585/
```

Alternatively you an go to the IBM Redbooks Web site at:

```
http://www.ibm.com/redbooks
```

Select the **Additional materials** link and open the directory that corresponds with the Redbook form number (SG246585).

Using the Web material

Download the **sg246585code.zip** file and use an unzip tool to expand it to your hard drive. Make sure you select the option to create the folder names. After unzipping the file, you should see the following directory structure:

```
Redbook
   sg246585
      sampcode
         application
         Chapter6
         Chapter7
         Chapter8
         Chapter11
         Chapter27
         Chapter28
         setup
```

Note: The sample code is shipped as individual files under each chapter directory. The \application sub-directory also contains a jar, war and ear file for the specific projects. The automatically generated files were not removed from the War and EAR file before they were created. If you want, you can use these files to load all the sample code into your workspace.

System requirements for downloading the Web material

There are no additional hardware requirements for downloading and using the Web material.

The sample code assumes that you have DB2 UDB database installed. The samples have been tested with Version 7.2 of DB2 UDB for Windows.

How to use the Web material

Instructions about how to use the samples are provided in the chapters where the individual sample is introduced and described.

Installing the sample database

The samples in this redbook are based on a DB2 database named ITSOWSAD. To install the database on your workstation, follow these steps:

Important: Make sure your user ID has the proper user rights to create a DB2 database and add tables to it.

1. Open a DB2 command window.
2. If the database manager is not already started, issue the **db2start** command.
3. Go to the sample directory:

   ```
   x:\Redbook\sg246585\sampcode\setup
   ```

Note: The following commands try to delete any existing objects so you may see a few errors as they are executed.

4. Run the command:

   – `db2 -tf itsowsad.ddl`

The expected output from this command is:

```
C:\REDBOOK\sg246585\sampcode\setup>db2 -tf itsowsad.ddl
SQL1024N  A database connection does not exist.  SQLSTATE=08003

DB20000I  The CREATE DATABASE command completed successfully.


   Database Connection Information

 Database server        = DB2/NT 7.2.1
 SQL authorization ID   = DB2ADMIN
 Local database alias   = ITSOWSAD


DB20000I  The SQL command completed successfully.
```

```
DB21034E  The command was processed as an SQL statement because it was not
valid Command Line Processor command.  During SQL processing it returned:
SQL0204N  "ITSO.AAPARTS" is an undefined name.  SQLSTATE=42704

DB21034E  The command was processed as an SQL statement because it was not
valid Command Line Processor command.  During SQL processing it returned:
SQL0204N  "ITSO.AAINVENTORY" is an undefined name.  SQLSTATE=42704

DB21034E  The command was processed as an SQL statement because it was not
valid Command Line Processor command.  During SQL processing it returned:
SQL0204N  "ITSO.MMPARTS" is an undefined name.  SQLSTATE=42704

DB21034E  The command was processed as an SQL statement because it was not
valid Command Line Processor command.  During SQL processing it returned:
SQL0204N  "ITSO.MMINVENTORY" is an undefined name.  SQLSTATE=42704

DB20000I  The SQL command completed successfully.

...
```

---

5. Run the command:

   – `db2 -tf itsowsad.sql`

The expected output from this command is:

```
C:\REDBOOK\sg246585\sampcode\setup>db2 -tf itsowsad.sql

   Database Connection Information

 Database server        = DB2/NT 7.2.1
 SQL authorization ID   = DB2ADMIN
 Local database alias   = ITSOWSAD


SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a
query is an empty table.  SQLSTATE=02000

SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a
query is an empty table.  SQLSTATE=02000

SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a
query is an empty table.  SQLSTATE=02000

SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a
query is an empty table.  SQLSTATE=02000

DB20000I  The SQL command completed successfully.
```

...

These commands will create the ITSOWSAD database along with these four sample tables:

- itso.aainventory
- itso.aaparts
- itso.mminventory
- itso.mmparts

Abbreviations and acronyms

AAT	application assembly tool
ACL	access control list
API	application programming interface
BLOB	binary large object
BMP	bean-managed persistence
CCF	Common Connector Framework
CICS	Customer Information Control System
CMP	container-managed persistence
CORBA	Component Object Request Broker Architecture
DBMS	database management system
DCOM	Distributed Component Object Model
DDL	data definition language
DLL	dynamic link library
DML	data manipulation language
DOM	document object model
DTD	document type description
EAB	Enterprise Access Builder
EAI	Enterprise Application Registration
EAR	enterprise archive
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
EJS	Enterprise Java Server
FTP	File Transfer Protocol
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMS	Information Management System
ITSO	International Technical Support Organization
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JAF	Java Activation Framework
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JFC	Java Foundation Classes
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSDK	Java Servlet Development Kit
JSP	JavaServer Page
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MFS	message format services
MVC	model-view-controller
OLT	object level trace
OMG	Object Management Group
OO	object oriented

© Copyright IBM Corp. 2002

OTS	object transaction service
RAD	rapid application development
RDBMS	relational database management system
RMI	Remote Method Invocation
SAX	Simple API for XML
SCCI	source control control interface
SCM	software configuration management
SCMS	source code management systems
SDK	Software Development Kit
SMR	Service Mapping Registry
SOAP	Simple Object Access Protocol (also known as Service Oriented Architecture Protocol)
SPB	Stored Procedure Builder
SQL	structured query language
SRP	Service Registry Proxy
SSL	secure socket layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UCM	Unified Change Management
UDB	Universal Database
UDDI	Universal Description, Discovery, and Integration
UI	user interface
UML	Unified Modeling Language
UOW	unit of work
URL	uniform resource locator
VCE	visual composition editor
VM	virtual machine
VXML	voice extensible markup language
WAR	Web application archive
WAS	WebSphere Application Server
WML	Wireless Markup Language
WS	Web service
WSBCC	WebSphere Business Components Composer
WSDL	Web Service Description Language
WSTK	Web Service Development Kit
WTE	WebSphere Test Environment
WWW	World Wide Web
XMI	XML metadata interchange
XML	eXtensible Markup Language
XSD	XML schema definition

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 684.

- *Web Services Wizardry with WebSphere Studio Application Developer,* SG24-6292
- *Self-Study Guide: WebSphere Studio Application Developer and Web Services*, SG24-6407
- *WebSphere Version 4 Application Development Handbook,* SG24-6134
- *WebSphere V4.0 Advanced Edition Handbook*, SG24-6176
- *Programming J2EE APIs with WebSphere Advanced*, SG24-6124
- *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144
- *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server*, SG24-5754
- *Programming with VisualAge for Java Version 3.5*, SG24-5264
- *WebSphere V3.5 Handbook*, SG24-6161
- *Version 3.5 Self Study Guide: VisualAge for Java and WebSphere Studio*, SG24-6136

Referenced Web sites

These Web sites are also relevant as further information sources:

- WebSphere Developer Domain

  `http://www7b.boulder.ibm.com/wsdd/`
- IBM Patterns for e-business Downloads

  `http://www.ibm.com/developerworks/patterns/install/index-lite.html`

© Copyright IBM Corp. 2002

- SCM Adaptor Vendors

  http://www.ibm.com/software/ad/studioappdev/partners/scm.html
- Eclipse

  http://www.eclipse.org/
- CVS

  http://www.cvs.org/, http://www.cvsnt.com
- JUnit

  http://www.junit.org/
- Ant

  http://jakarta.apache.org/ant
- Apache Ant 1.5 Manual

  http://jakarta.apache.org/ant/manual
- Xalan - Java Version 2.4D1

  http://xml.apache.org/xalan-j
- Trail: Writing Applets

  http://java.sun.com/docs/books/tutorial/applet/index.html
- SOURCEFORGE.net

  http://vaj2cvs.sourceforge.net
- Java Virtual Machine Profiler Interface (JVMPI)

  http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html
- Concurrent Version Systems

  http://www.cvshome.org
- Rational ClearCase

  http://www.rational.com/products/clearcase
- Article: “Using Ant with WebSphere Studio Application Developer”

  http://www7b.boulder.ibm.com/wsdd/library/techarticles/0203_searle/searle1.html

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

Symbols
/Open SQL 262

A
Action 173
Activity 564
Add Bookmark 50
Add Child 58, 80
Add External JARs 152
Add Filter 477
Add Import 49, 125
Add Java Exception Breakpoint 93
Admin Console 421
Administrative tools 474
Administrator's Console 421
AEs 18
AEs Test Environment 74
Agent Controller 479
All Extensions 608
Allow termination of remote VM 402
Animated GIF Designer 13, 54
Ant 442, 626
Ant build files 442
Ant build path 443
Ant build project 442
Ant build property 443
Ant build target 442
Ant build task 442
Ant Console 635
Ant execution log 447
Ant Extra Plug-in 450
Ant messages 447
ant.jar 630
AntTask 625–626, 631
Apache Tomcat 18, 74, 365, 392
appClientExport 451
Application Client Modules 62
Application client project 115
Application client project organization 120
Application Development Handbook 552
Application editor 63
Application Extension Editor 66
Application Server 11
Archives 181
Assertions 408
Attaching a Java process 475
Author 516
Automatic Builds 408
Automatic compilation 160

B
Back-end transaction servers 339
Backing up the CVS repository 550
Base set of objects 467
Base time 461
Base Version 490
Baseline 565
Bean page 69
Bindings page 73
BMP 196
Bookmark 50, 134–135
Bootstrap port 671
Branch 495
Branching 546
Breakpoint 157, 395
Breakpoint view 92
Build 30
Build files 442
Build output folder 143
Build paths 160
Build scripts 552
Build targets 446
build.properties 649
build.xml 444–445
Built-in tasks 443
Business logic 217
BusinessObject 354
By feature 105
By type 105

C
CardQuery.java 345
Cascading Style Sheet 163, 198
Catch up 103, 489, 533, 541
CGI 173
Change Variable Value 400

© Copyright IBM Corp. 2002

Changeable scope 49
Check in/Check out 565
Checkouts in local mode 565
Checkouts in server mode 565
Child Resource 38
Class bytecodes 160
Class path 24, 345
Class Statistics 95, 460
CLASSPATH 75, 391
Classpath Variables 24
ClearCase 159
ClearCase administrator 568
ClearCase Check In 579
ClearCase Check Out 580
ClearCase Explorer 582
ClearCase LT 16, 101, 494, 556
ClearCase MultiSite 558
ClearQuest 557
Client-centric approach 222
Client-server access 498
cm_status.bat 561
Code Assist 49, 123
Code Base 181
Code Formatter 25
Code formatting 48–49
Collected 462
Color by 463
Colors view 54
COM.ibm.db2.jdbc.app.DB2Driver 302
COM.ibm.db2.jdbc.net.DB2Driver 302
Common Public License 5
Compare two versions 528
Compare with 485
Comparing resources 528–529
Comparing with repository versions 530
Compilation target 446
Compile errors 156
Completed 107
Component 564, 651
Component ID 653
Component Version 653
Concurrent Versions System 16, 101, 494, 498
Conditions 287, 293
Configuration type 384
Conflicting changes 533
Connect to Rational ClearCase 570
Connection pooling 263
Connection timeout 381
Connection timeout field 434
Consistency 407
Console view 45
Consolidate Roles 65
Constructors from superclass 208, 226
Container page 73
Content assist 49, 56
Context menu options 56
Context root 65, 163
Control visibility of ancestor pane 544
Controller 219
Controller servlet 313
Conventional performance tools 94
Convert links to document relative links 195
Copy current change from right to left 545
Copy file transfer mechanism 378, 432
Copy Version To Stream 520
Core Resource Management 630
Create a CSS File 199
Create a database 276
Create a JSP File 231
Create a New SQL Statement wizard 284
Create a new table 278
Create a Stream 522
Create Component JAR 657
Create directory structure for files 155
Create fragment JARs 648
Create groups 670
Create New Folder 143
Create plug-in JARs 638
Create users 670
Create VOB 568
Create Web Page from an Java bean 183
Creating a component 650
Creating the test case 410
CSS File Wizard 198
Ctrl+Shift+M 49
Ctrl-H 133
Ctrl-Space 49, 124
Cumulative Time 462
Custom taglibs 246
Custom tags 234
CVS 159
CVS commands (other) 553
CVS Ignore 550
CVS readers and writers 551
CVS Repository Location 505
CVS tags 512
CVS Web site 502
CVSNT 502

D

dab
  getColumn 320
  parameter 320
  repeat 320
DAD 84
DADX 78
Data model 219
Data Perspective 266
Data source 263, 309
Data view 90, 268
Database connection 270
Database wizard 59
Databases 62
DB Beans 303, 315
DB Explorer 88, 267, 269
DB Explorer view 86
DB2 180, 229
DB2 App Driver 271
DB2 driver location 377, 429
DB2 JDBC Driver 151
DB2 Net Driver 271
DB2 table 302
DB2 XML Extender 78, 85
db2java.zip 25, 303
Db2JdbcDriver 264
dbbeans.jar 315
dbbeans_javadoc.zip 315
DBConnectionSpec 316
DBProcedureCall 317
DBSelect 316
DDL 85–86, 265
Debug perspective 91
Debugging 157
Debugging JSPs 394
defailt_app.webapp 348
Default Application Server 671
Default Perspective 39
Default Server Web container 674
Default Text Editor 274
Define the Servlet in the Deployment 209
Deliver activities 583
Deploy the application 426
Deprecated 347
Design Patterns 406
Design view 54–55
Details Form 305
Developer Resource Portal 12
Disconnect 403
DOCTYPE 58
Document root 118
DriverManager 302
driverManagerSpec 320
DRP 12
DTD 14, 57, 76, 83, 166
DTD editor 77, 80
Dump Object Reference 468
Dynamic and snapshot views 558

E

EAR 64, 112, 163, 326, 443, 450
EAR file 326
EAR file export 420
earExport 451–452
Editing JSP 233
Editors 35
EJB 64, 355
EJB Bindings page 67
EJB Deploy 450
EJB editor 68
EJB Extension editor 72
EJB JARs 450
EJB module 62, 69
EJB project 115, 364
EJB to JNDI Name Mapping 423
EJB1.1 specification 334
EJBCommandTarget 328
ejbDeploy 450
ejbExport 450
EJBModule 115
ejbModule folder 62
Element selection 56
Enable profile server process 472
End State page 525
Enterprise 12
Enterprise application archive 16
Enterprise Application project 114, 364
Enterprise Applications 62, 422
Enterprise Bean Java Editor 69
Enterprise Developer 12
Enterprise Generation Language 12
Enterprise project organization 117
Environment 473
Environment page 70
Error Log 595
Error Page 305
Execution flow 95–96

Execution Flow view 468
Export 154
Export source files 421
Exporting an EAR 452
Extensions page 67
Extention points 596
External JAR files 48
External plug-ins 591
eXtreme Programming (XP) software 406

F

Finders page 73
Firewall Settings 381, 434
FORM 173
Form and Input Fields 174
Format 26, 177
Fragment development 642
Fragment project 643
FTP file transfer 432
FTP file transfer mechanism 378, 434
FTP server 426
FTP URL field 434

G

Gallery of images 196
Gallery view 53
Garbage collection 465
Garbage collection statistics 459
Garbage collection thread 469
Gather Roles from Modules 65
Generate DDL 281
Generate DDL files and XML Schemas 274
Generate DDL from an XMI 266
Generate debug information when compiling JSPs 394
Generate new Java class 611
Generate plug-in configuration file 390
Generate XML Schema 275
Generic Wizards 609
getConnection 303
GIF 196
Global properties 446
GROUP BY 288

H

HEAD 489, 512, 514, 542, 548
Heap 95, 459, 463
Heap view 463
Hide Editors 36
Hierarchy 38
Hierarchy view 45
HomeImpl class 354
Host 402
Host address 376, 428
HTML 52
HTML editor 232
HTML tags 351
HTTP Post 173
HTTP request 184
HttpServletCommandServer 329
HttpSession 347

I

IBM Agent Controller 92, 365, 390, 426
IBM Data access bean library 346
IBM DB2 XML Extender 15
IBM HTTP server 391
IDE 34
IDE Remote Tool 358
Ignore (CVS) 550
Ignored Resources 550
ImageURL 239
Import assistance 49
Importing an existing Web site 193
In Folder 107
Incoming changes 533
Inheritance page 72
Inherited abstract methods 208, 226
Initial State page 524
Initialization target 446
Inner classes 347
Input Form 305
Insert Bean 235
Insert Custom 319
Insert scriplet 243
Insert Table 238
Inspect 123
Inspector 92
Installed JREs 29
Installing the EAR 421
Instance Name 374
Integrated debugging 136
Integrated debugging features 49
Integration Edtion 11
Internal and External Editor 620

Internet Explorer 664
Invoker 350
ISelectionListener 617
ITSOJarEditor 619, 646
ITSOJarView 612
ITSOWSAD 264, 267, 285, 307
ITSOWSADTeam 540
ItsoWsDealerInstanceCopy 374
ItsoWsDealerInstanceFTP 374
ItsoWsDealerParts 142
ItsoWsDealerWebProject 205, 247, 396, 451
ItsoWsDealerWebProjectEAR 452
ivjdab.jar 346

J

J2EE 165, 450
J2EE hierarchy 112
J2EE perspective 61
J2EE view 62
Jakarta 442
JAR 48, 114, 144
Java 25
Java 2 platform 340
Java Applet 177
Java Application 150, 348
Java Attribute Page Wizard 611
Java Build Path 151, 186, 450
Java Build settings 142, 164
Java Builder 127
Java Code Editor 48
Java development tooling 590
Java Development Tools Core 630
Java Editor 25–26, 49, 124–125
Java elements 600
Java Native Directory Interface 263
Java perspective 43
Java project 113, 140
Java project organization 118
Java search 52, 133
Java snippets 122
Java utility JAR 450
Java Virtual Machine 458
Java Virtual Machine Profiler Interface 459
JavaBean wizard 59
JavaBeans 182, 234
JavaScript 13
JavaServer Pages 9
Java-wrapper classes 306
JDBC 25, 151, 262, 302
JDBC 2.0 Standard Extension API 263
JDBC driver 24
JDBC driver class 271
JDK 122, 160
JDT 625
JDT API 599
JIT compiler 472
JNDI 73
JNDI name 265
Join 286
JPEG 196
JRE 29, 160
JRE JAR file 30
JRE library 144
JRE source file 30
JSP 11
JSP Declaration 237
JSP Directive - taglib 318
JSP Expression 244
JSP SQL 317
JSP taglib model 59
JSP tags 351
JSP Wizards 59
jspsql.jar 317
JUnit 406
junit.framework.Assert 413
JUnit.framework.TestCase 409
junit37src.jar 410
JViewport 347
JVMPI 459

K

Keep checked out 576, 579

L

Language 244
Libraries 151
Life-cycle 628
Link Insertion Wizard 175
Link Navigator Selection to Active Editor 49
Links view 54
Listing 147
ListParts 291
Local and remote Servers 363
Local history 484
Local repository 511
Location Service Daemon 671

Lock View icon 47

M

Maintenance 408
Maintenance stream 547
MAKE 442
Manifest editor 594
Manually Debugging 401
Matching types 186, 208
Memory intensive classes 459
Memory leaks 465
Memory-intensive classes 464
Merge 523
Merge versions 583
Merge view 526
Merging 534
Merging from a stream 523
Merging streams 548
META-INF 62
Method execution 95
Method invocation 95
Method statistics 95, 462
Methods page 72
MIF 196
MMINVENTORY tables 285
MMPARTS 284, 290–291
Model 219, 225, 252, 305
Model synchronized 222
Model-View-Controller 218
Move Project Into ClearCase 575
Multiple JVMs 458
Multiple workspaces 486
MVC based Web application 257
MVC pattern 59, 222
MVC sample 20
MVC usage rules 220

N

Navigator view 45, 62, 277, 486
New Connection... 270
New Filter 272, 477
New Stream dialog 514
New-generation objects 468
Non-Java resource 118

O

Object Management Group 115, 266
Object Reference view 466
Object references 95
Observer of the Model 219
ODBC 262
Offline 483
OLE 39
On selected resource only 157
Open Perspective 40, 94, 398
Open Source 468
Optimistic concurrency model 487, 491
ORDER BY 288
Organize Imports 126
Outgoing changes 533
Outline view 45, 53, 56–57, 63, 68, 124
Overwrite existing files 421

P

Package 146
Packages view 44
Page Designer 54, 166, 181, 230, 245
Parent Resource 38
Parser time 627
PartList 213
PartList servlet 410
PartList.html 171
PartListApplet 178
PartListBean 183, 412
PartListBeanSP 300
PartListBeanTester 411
PartListBeanViewBean 192
Passive Mode 381, 434
pathvar 633
PDE Runtime 596
PDK Command Servers 327
PDKLite 327, 332
Performance profiling 458
Persistence Builder 354
Perspective menu 36
Perspectives 34
Platform Extensions 608
Platform subsystem 590
Pluggable JDK 122
Pluggable third party repository 493
Plug-in Development 593
Plug-in Development Environment 18, 589–590
Plug-in fragment 642
Plug-in JAR 638
Plug-in manifest editor 594

Plug-in Project 604
Plug-in Registry 595
Plug-in runtime library 605
plugin-cfg.xml 390
PNG 196
Port 402
Port 2401 503
POST 416
Predicate 272
Preferences 31, 126, 592
Presentation logic 217
Preview 59
primaryKey 334
Priority 107
Process ID 474
Processes view 92
Profiling Perspective 93
Project 113
Project class path 628
Project folder field 379
Project structure 340
Project version 520
Properties and method parameters 188
Properties view 53, 58, 63
Publishing a JAR 658
Publishing a plug-in 638
PVCS 558

Q

Query Applet 19
Query Application 19
Quick links 72

R

Ramp-up time 408
RDB to XML mapping editor 78, 84
RDB_node mapping DAD 84
Rebase 565
recycle bin 100
Redbooks Web site 684
  Contact us xxii
Redbooks.gif 171
Refactoring 28, 48, 128
References between instances 467
References page 71
Relationships page 73
Release 103, 489, 539
Release a Project 536
Release mode 537
Remote Agent Controller 480
Remote file transfer instance 430
Remote file transfer name field 379
Remote repository 512
Remote server 426
Remote target directory 434
Remote target directory field 379–380, 434
Remote WebSphere 401
RemoteException 334
Repetition of an object 468
Repositories view 102, 500
Repository 488
Repository Location 507
Repository management 551
Repository path 509
Request 184
Resolve conflicts 537
Resource History 500
Resource History View 39
Resource perspective 37
Restore Default Values 58
Resume 399
Retrieve 177
RMIC 450
Role to User Mapping 423
Role-based development model 4
Run 149
Run Ant... 447, 634
Run as mode 72
Run As Role 67
Run ClearCase Explorer 582
Run on Server 59, 76, 191, 214, 254, 315, 348, 367, 396, 437
Run time 628
Run to Return 399
Runtime-workspace 592, 614

S

Save all modified resources 129
Scenario
  parallel development 542
Schema-Based Extensions 609, 646
SCM 159, 483
SCM adapters 494
SCM integration 493
SCM perspectives 499
Scope 236

Scrapbook 99, 122
Scriplet 237
Script Perspective 96
SEAppInstall 421
Search 52, 133
Search view 45
Security (CVS) 551
Security Bindings page 67
Security page 70
SELECT 284, 290, 316
Sequential development scenario 539
Server 372
Server centric approach 223
Server Configuration 62, 75, 383
Server Control Panel 75
Server Instance 62, 372
Server Instance and Configuration 386, 391, 427
Server Instances folder 381
Server Perspective 366, 396
Server project 116
Server project organization 120
Server Tools 363–364
server.xml 120
server-cfg.xml 120, 390, 552
Servlet 11, 204
Servlet wizard 205
Session 184
Set a breakpoint 395
Set as default launcher 150
Set Default 39
setAttribute 348
setUp and tearDown methods 412
SG245264.jar 342, 344, 349
SG24-5264EAR 344
Shared repository 487
Shortcut 172
Show Caller 465
Show Debug Perspective 150
Show Editors 36
Show Method Invocation 465
Show the history 103
Show Version Info 521
Show View 36, 42
Simple Build file 444
Simple Object Access Protocol 16
Simple project 113
Simple WAR Configuration 387
Simple WAR configuration 384
SimpleServlet 251, 253
SimpleServlet.java 210
Single server edition 363
Size 462
Smalltalk GUI library 218
Snapshot views 558
Software Configuration Management 483, 498, 556
Source 56
Source Code Management plug-ins 340
Source Compare pane 538
Source folder 605
Source page 71
Source view 54–55
Split a stream 519
Splitting based on a project version 519
Splitting with changes in your workspace 522
SQL query 304
SQL Query Builder 89–90, 290
SQL statement 123, 284, 307
SQL Wizard 89, 284
SQL wizard 290, 308
SQLJ 346, 354
Start Admin Server 421
Start Monitoring 478
Static field (EJB) 334
Step Into 399
Step Over 399–400
Stopping remote instance 392
Stored Procedure 20
Stored procedure 298
Stored Procedure Builder 298
Storing source code 159
Stream 488, 495, 512, 565
Stream Contents 530
Structure Compare pane 533, 537
Structured Query Language Java 354
Style 173
Style Sheet 305
Styles view 54–55
Submit Button 174
Superinterfaces Selection 208
Supertype hierarchy 47
Suspend 399
Switching to/from ClearCase 561
SWT 590
Synchronize 103, 106, 489, 492, 537
Synchronize view 500
Syntax highlighting 49, 56

T

Table 238
Taglib 305
Target 272
Tasks view 45, 63, 106, 127
Team development 487
Team roles 488
Team specific actions 527
Template 375
Template wizards 606
Terminate 399
TestRunner 414
TestSuite 414
Text Field 174
Text search 52
Theme 119
Threads 470
Three way compare 528
Thumbnail 170
Thumbnail view 54
Time consuming classes 459
Time consuming objects and methods 464
Tomcat 120
Transaction page 70
Tutorial 563
Type Hierarchy View 45

U

UCM 564
UDB 156
Unified Change Management 556
Unit test cases 407
Universal Description Discovery and Integration 16
UNIX 474
Unlimited undo 56
Unreserved check-out 498
Update Views 480
URI 65
URL 303
URN 65
Use an existing remote file transfer instance 432
Use default library 30
Use Default Location 371
Use default WebSphere deployment directory 389, 429
Use existing database model 285
Use Existing SQL Statement 307
Use Firewall 434
Use PASV Mode 381, 434
Use Single Thread Model 207
useBean 303
user interface (UI) 5
User-defined macros 56
utilJar 451

V

VAJ2CVS 357
Variables views 92
VBScript 13
VCE 178, 345
Version 490
Version Conflict 543
Version from Stream 547
Version from stream 535
Version From Workspace 522
Version from workspace 535
Version label 536
Versioning 490
View 34, 219, 252
View Bean 59, 184
View Bean wrapper 190, 314
View-tag 564
Visual Composition Editor 178, 341
VisualAge for Java 19, 140, 159, 178, 340, 354
VisualAge for Java Libraries 24
VisualAge for Java XML tools 340
VOB 495, 564, 568

W

W3C 82
WAR 59, 112, 450
warExport 451
Watching a file for changes 553
Watching variables 400
Web 165
Web application 363
Web application archive 16
Web Art Designer 13
Web Browser 59, 166
Web Modules 62
Web pages from SQL queries 304
Web perspective 52
Web project 114, 162, 207, 364
Web project organization 118
Web resource 118
Web service 11

Web Services Description Language 16
web.xml 59, 166, 248, 336, 349
WebApplication 119, 165, 168
WebArt Designer 54
WEB-INF 348
WebSphere Admin Console 421
WebSphere Application Server 204, 364
WebSphere Application Server Attention 672
WebSphere deployment directory 376, 429, 433
WebSphere deployment directory field 379
WebSphere installation directory 376, 429
WebSphere Remote Server 427
WebSphere Studio Advanced Edition V3 19
WebSphere Studio Asset Analyzer 12
WebSphere Studio Page Detailer 458
WebSphere Test Environment 348, 365
WebSphere V4.0 Test Environment 369
WHERE clause 90, 287, 292
Windows 2000 Server 670
Wizards 204
Work flow 492
Workbench architecture 5
Workspace 483–484
Workspace plug-ins 591
WSAA 12
wsappdev 487
WSCP 552
WYSIWYG 54–55

X

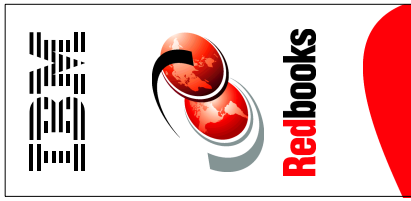
XMI 266
XML 60, 65, 442
XML and SQL query wizard 78
XML editor 77–78
XML Perspective 76
XML schema editor 77, 81
XML Schema Recommendation Specification 82
XML to XML mapping editor 78, 83
XSD 83
XSL trace editor 78, 83
XSLT 15

Y

Yellow arrows 133

Z

Zoom In/out 468



WebSphere Studio Application Developer Programming Guide

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

WebSphere Studio Application Developer
Programming Guide



Develop, test, build, and deploy your Web applications and plug-ins

Migrate your VisualAge for Java applications

Experience team programming concepts

This IBM Redbook is a programming guide for the application development tool, WebSphere Studio Application Developer. This tool is not only for the Java developer, but also for the Web designer who creates Web pages. The WebSphere Studio Application Developer basic tooling and team environment is presented along with the development and deployment of Web applications.

WebSphere Studio Application Developer provides integrated development tools for all e-business development roles, including Web developers, Java developers, business analysts, architects, and enterprise programmers. The customizable, targeted, role-based approach of WebSphere Studio Application Developer will be characteristic of all new products built on the WebSphere Studio Workbench. It is well integrated with WebSphere Application Server and provides a built-in single server that can be used for the testing and profiling of Web applications. It replaces the existing Java and Web application development tools, VisualAge for Java and WebSphere Studio.

This redbook consists of eight parts: an introduction to the WebSphere Studio family of products and a sample application that is used in this book, Web page and Java Web application development, database connectivity, migrating from VisualAge for Java, testing and deploying a Web application with WebSphere Application Server, profiling, team development with Concurrent Versions System or Rational ClearCase LT, and plug-in development in WebSphere Studio Application Developer.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6585-00 ISBN 0738425826